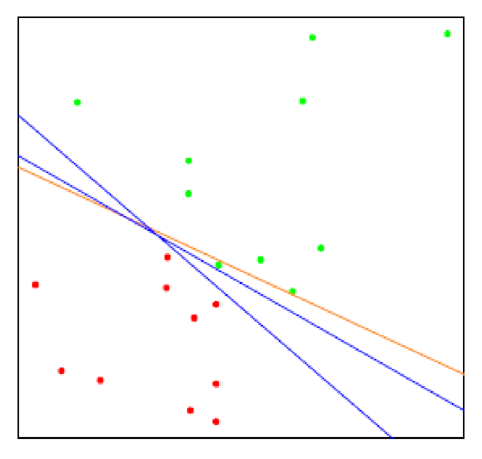


Deep Learning

Introduction to Deep Learning Models

Dr. – Ing. Morris Riedel

Adjunct Associated Professor

School of Engineering and Natural Sciences, University of Iceland

Research Group Leader, Juelich Supercomputing Centre, Germany

LECTURE 6

Fundamentals of Long Short-Term Memory

June 7th, 2018

Juelich Supercomputing Centre, Germany

UNIVERSITY OF ICELAND

SCHOOL OF ENGINEERING AND NATURAL SCIENCES

FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

JÜLICH Forschungszentrum

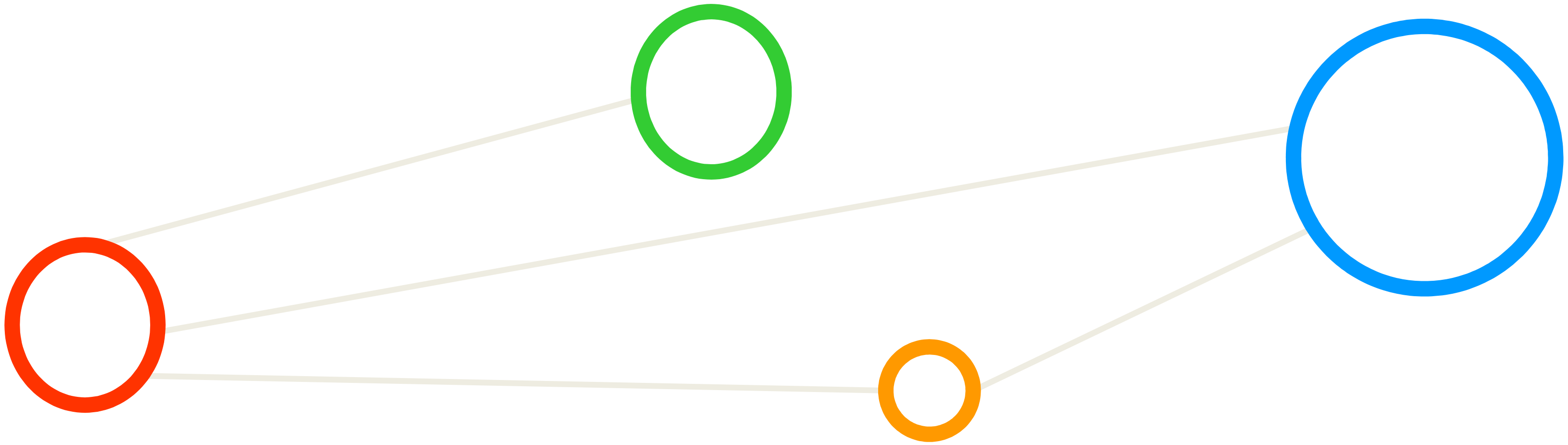
HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

DEEP Projects

Outline of the Course

1. Introduction to Deep Learning
2. Fundamentals of Convolutional Neural Networks (CNNs)
3. Deep Learning in Remote Sensing: Challenges
4. Deep Learning in Remote Sensing: Applications
5. Model Selection and Regularization
6. Fundamentals of Long Short-Term Memory (LSTM)
7. LSTM Applications and Challenges
8. Deep Reinforcement Learning

Outline

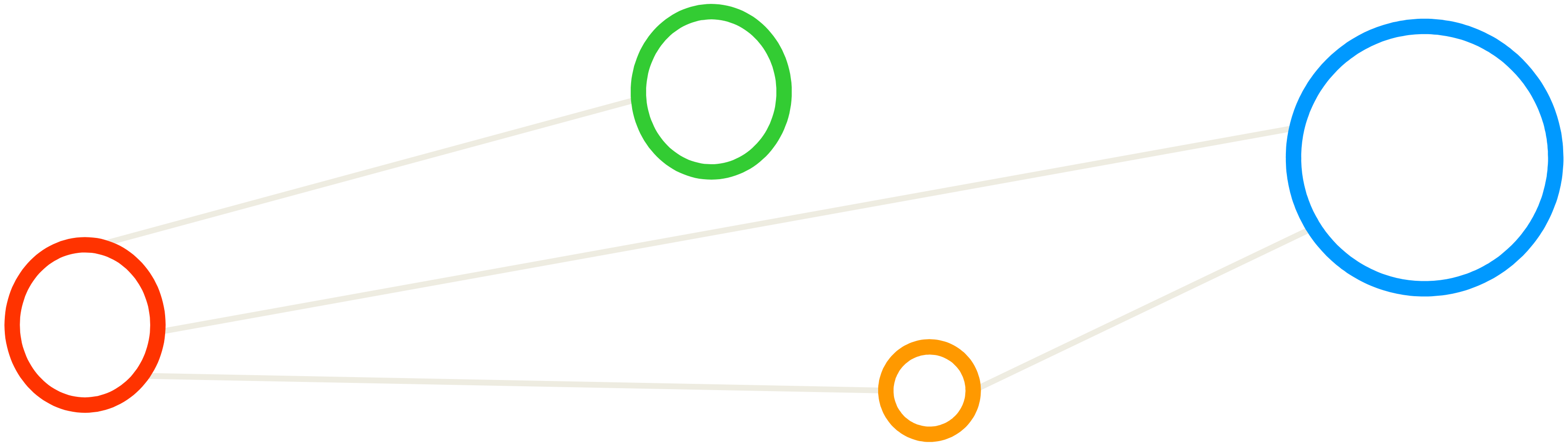


Outline

- Recurrent Neural Networks (RNNs)
  - Sequence Models & Dataset Impact
  - Limitations of Feed Forward Networks
  - RNN Model & Unrolling
  - RNN Cells & Topologies
  - Simple Application Example

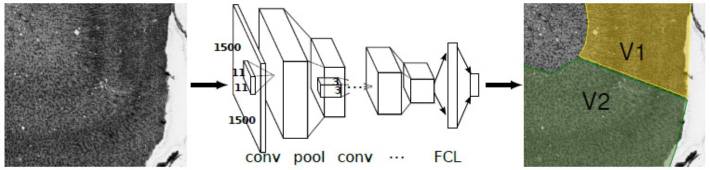
- Long Short-Term Memory (LSTMs)
  - LSTM Model & Memory Cells
  - Vanishing Gradient Problem
  - Keras and Tensorflow Tools
  - Different Useful LSTM Models
  - Simple Application Example

Recurrent Neural Networks (RNNs)



Deep Learning Architectures

- Deep Neural Network (DNN)
  - ‘Shallow ANN‘ approach with many hidden layers between input/output
- Convolutional Neural Network (CNN, sometimes ConvNet)
  - Connectivity pattern between neurons is like animal visual cortex

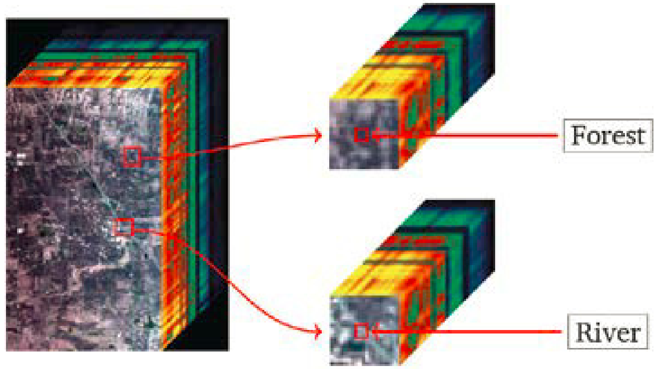


- Deep Belief Network (DBN)
  - Composed of mult iple layers of variables; only connections between layers
- Recurrent Neural Network (RNN)
  - ‘ANN‘ but connections form a directed cycle; state and temporal behaviour

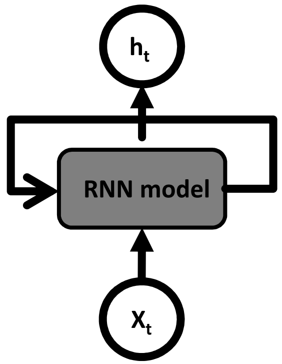
- **Deep Learning architectures can be classified into Deep Neural Networks, Convolutional Neural Networks, Deep Belief Networks, and Recurrent Neural Networks all with unique characteristica**
- **Deep Learning needs ‘big data‘ to work well & for high accuracy – works not well on sparse data**

Revisit CNNs vs. RNNs

- CNNs (cf. day one)
  - Example: remote sensing application domain, hyperspectral datasets
  - Neural network key property: exploit spatial geometry of inputs
  - Approach: Apply convolution & pooling (height x width x feature) dimensions



- RNNs
  - Examples: texts, speech, time series datasets
  - Neural network key property: exploit sequential nature of inputs
  - Approach: Train a graph of 'RNN cells' & each cell performs the same operation on every element in the given sequence



- **RNNs are used to create sequence models whereby the occurrence of an element in the sequence (e.g. text, speech, time series) is dependent on the elements that appeared before it**

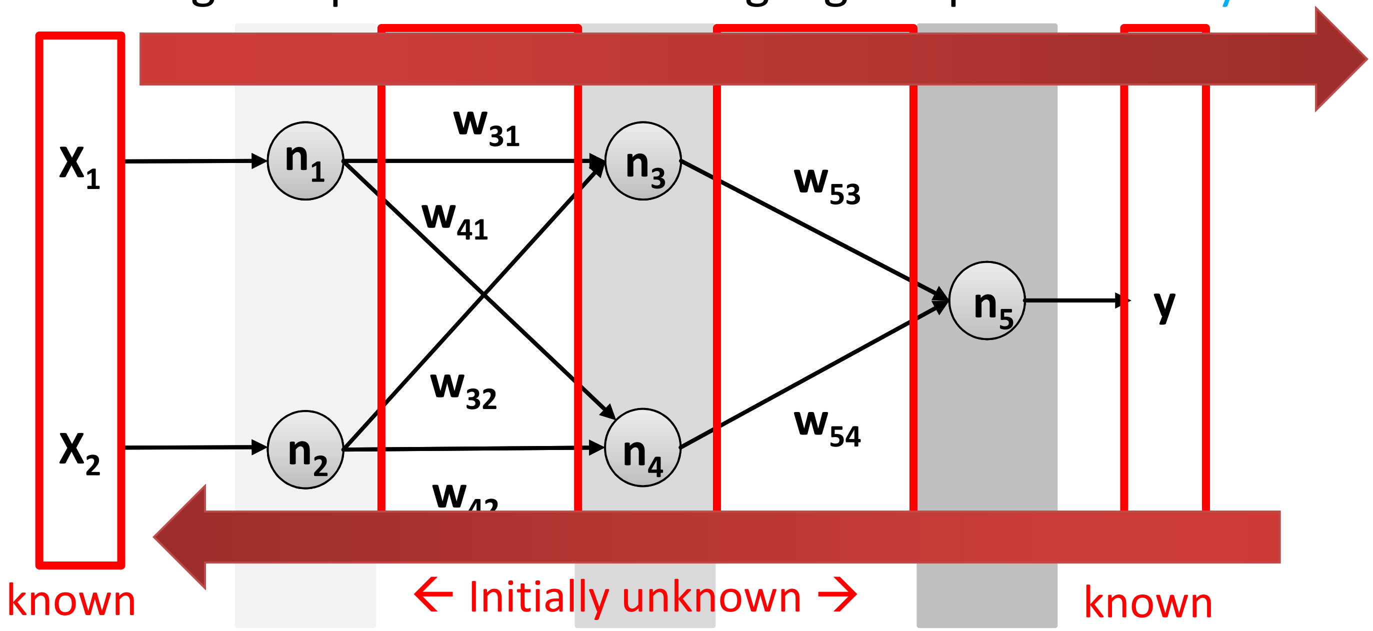
Sequence Models

- **Sequence models enable various sequence predictions that are inherent different to other more traditional predictive modeling techniques or supervised learning approaches**
- **In contrast to mathematical sets often used, the 'sequence' model imposes an explicit order on the input/output data that needs to be preserved in training and/or inference**
- **Sequence models are driven by application goals and include sequence prediction, sequence classification, sequence generation, and sequence-to-sequence prediction**

- Model Categorization
  - Based on different inputs/outputs to/from the sequence models
- Practical 'standard dataset' perspective
  - Often the order of samples is not important
  - Training/testing datasets and their samples have often no explicit order (i.e. 'sets')
- Practical 'sequence dataset' perspective
  - Order of samples is important
  - Sequence model learning/inference needs this order

Limitations of Feed Forward ANN (cf. Day One)

- Selected application examples revisited
  - Predicting next word in a sentence requires 'history' of previous words
  - Translating european in chinese language requires 'history' of context

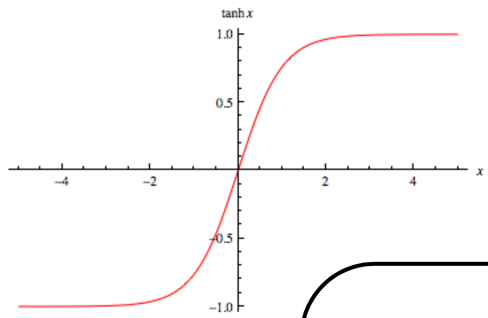


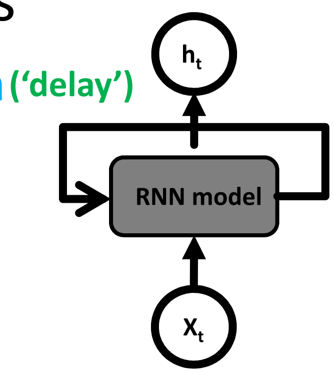
- **Traditional feed forward artificial neural networks show limits when a certain 'history' is required**
- **Each Backpropagation forward/backward pass starts a new pass independently from pass before**
- **The 'history' in the data is often a specific type of 'sequence' that required another approach**

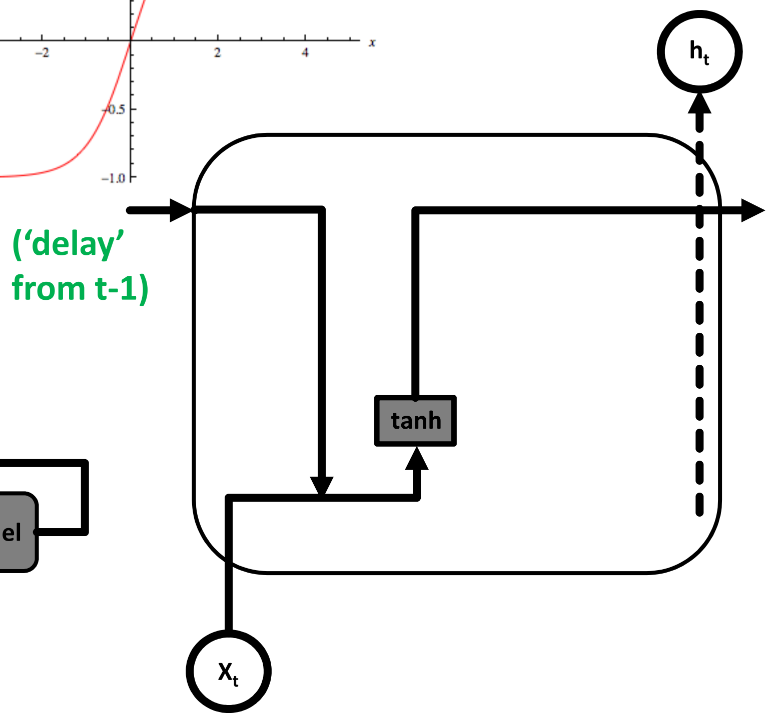
Recurrent Neural Network (RNN)

- **A Recurrent Neural Network (RNN) consists of cyclic connections that enable the neural network to better model sequence data compared to a traditional feed forward artificial neural network (ANN)**
- **RNNs consists of 'loops' (i.e. cyclic connections) that allow for information to persist while training**
- **The repeating RNN model structure is very simple whereby each has only a single layer (e.g. tanh)**

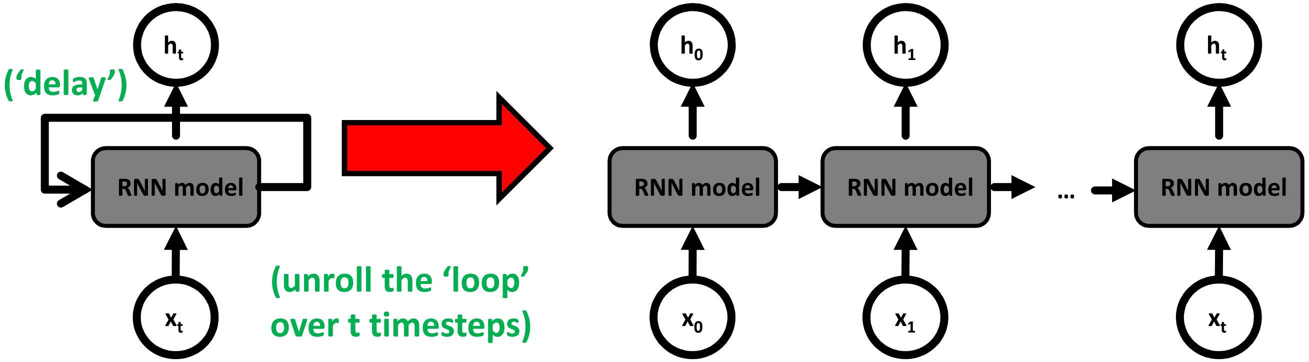
- Selected applications
  - Sequence labeling
  - Sequence prediction tasks
  - E.g. handwriting recognition
  - E.g. language modeling
- Loops / cyclic connections
  - Enable to pass information from one step to the next iteration
  - Remember 'short-term' data dependencies





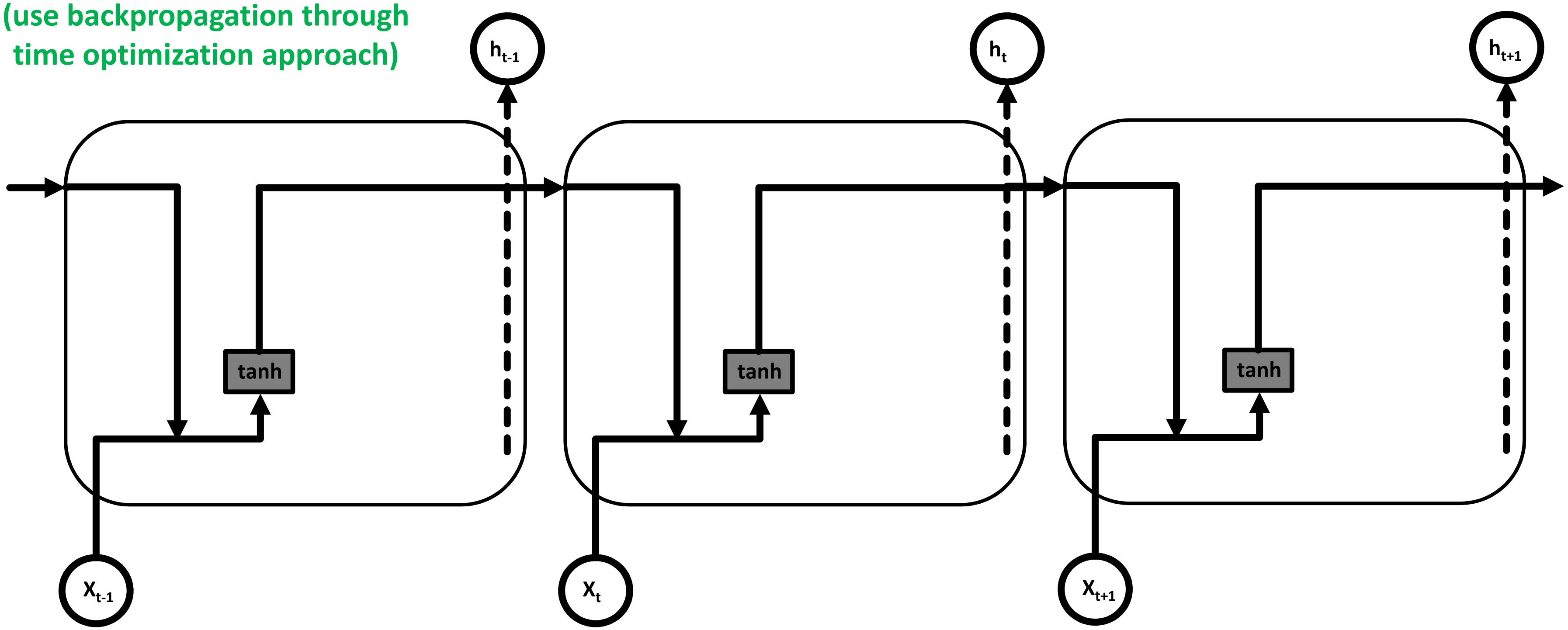


Unrolled RNN

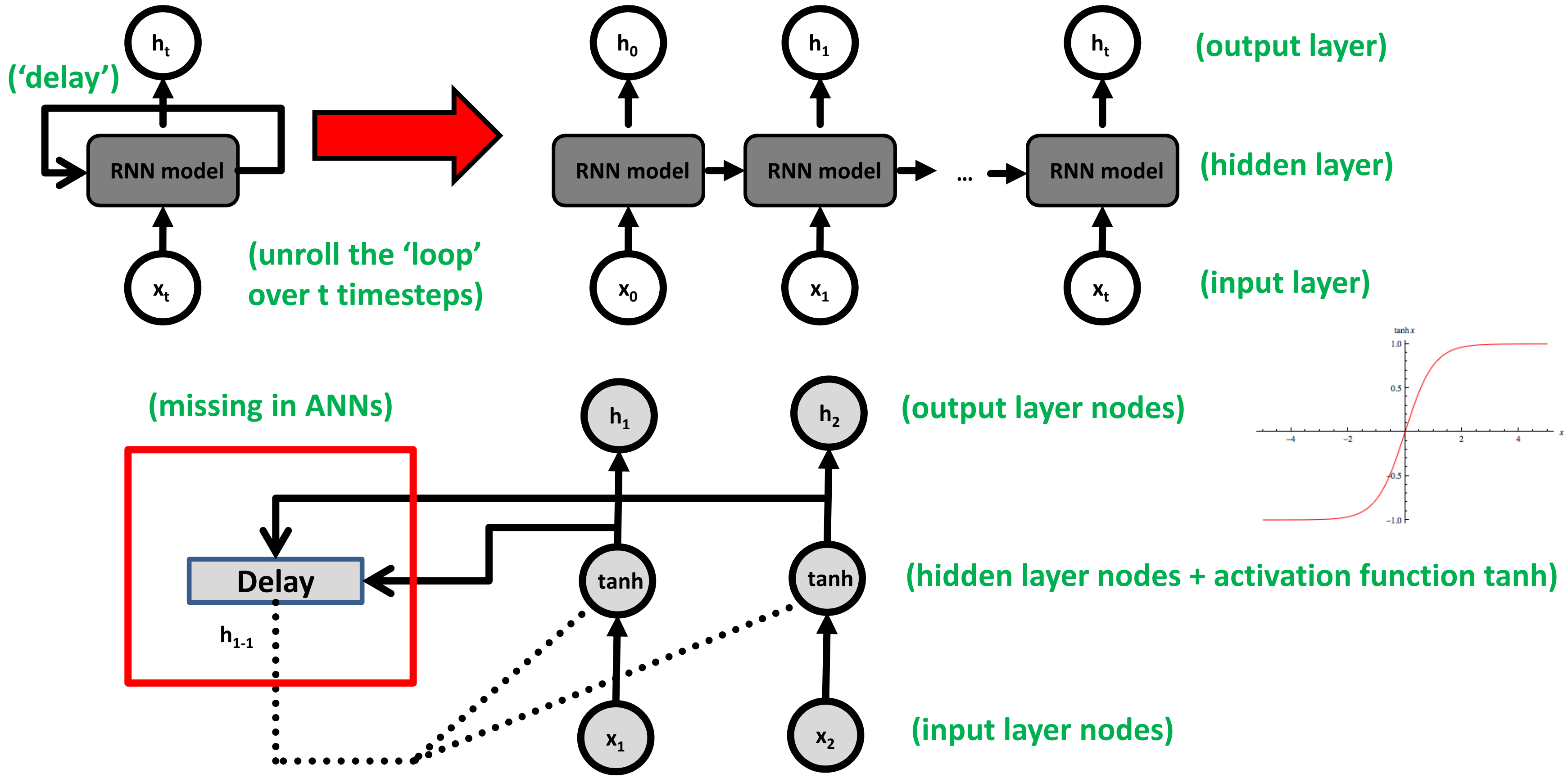


- A RNN can be viewed as multiple copies of the same network, each passing a message to a successor – this gets clear when 'unrolling the RNN loop'

(use backpropagation through time optimization approach)

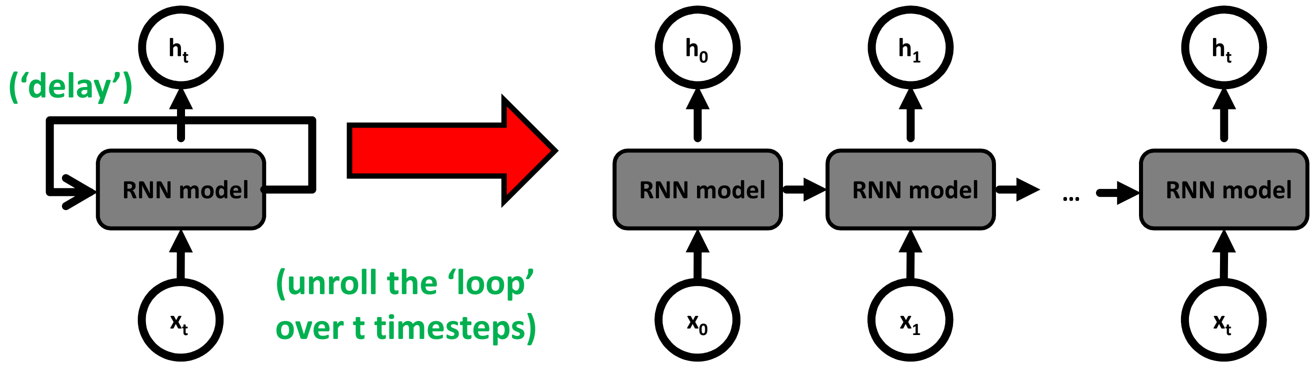


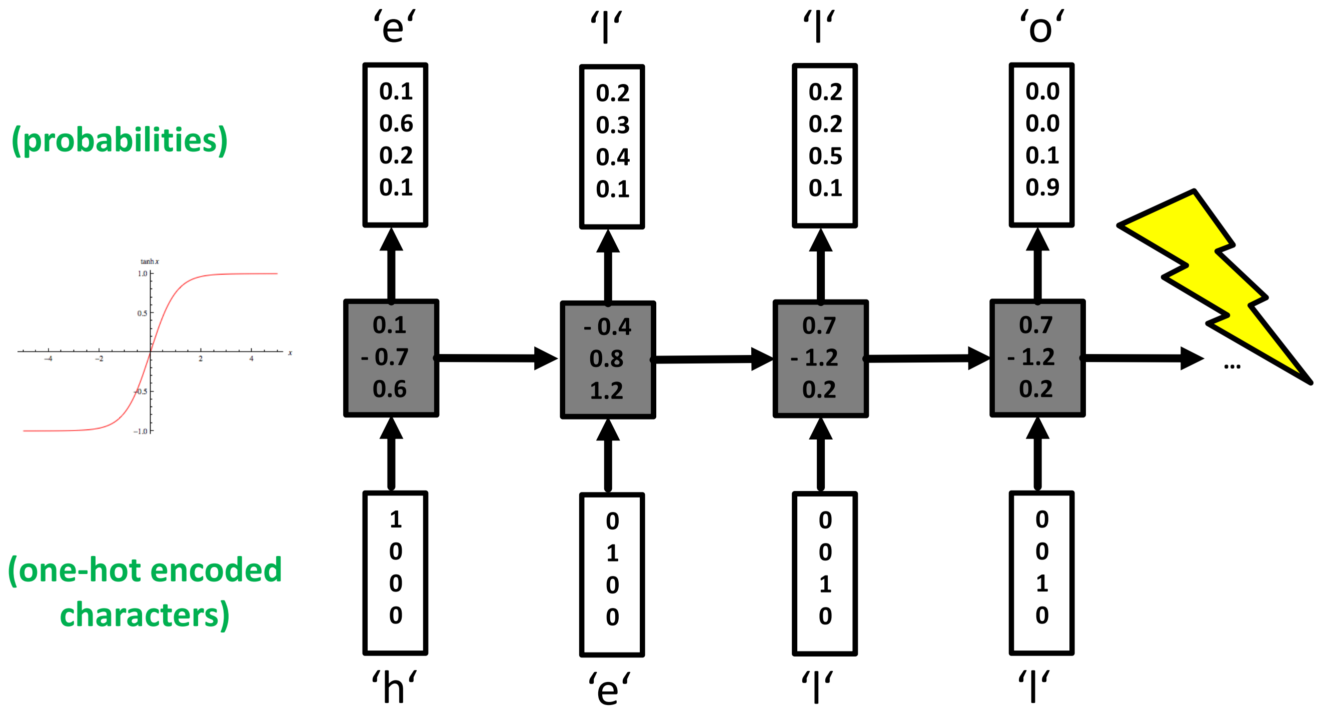
Unrolled RNN – Role of ‘Delay‘ and Nodes in Layers



- RNNs are unrolled programmatically during the training and prediction phase
- Idea of ‘delay‘ means feeding back the output of a neural network layer at a specific time t to the input of the same nerual network layer at time t+1 → establishes something like ‘short memory‘

RNN Model – Simple Example – Predict Next Character

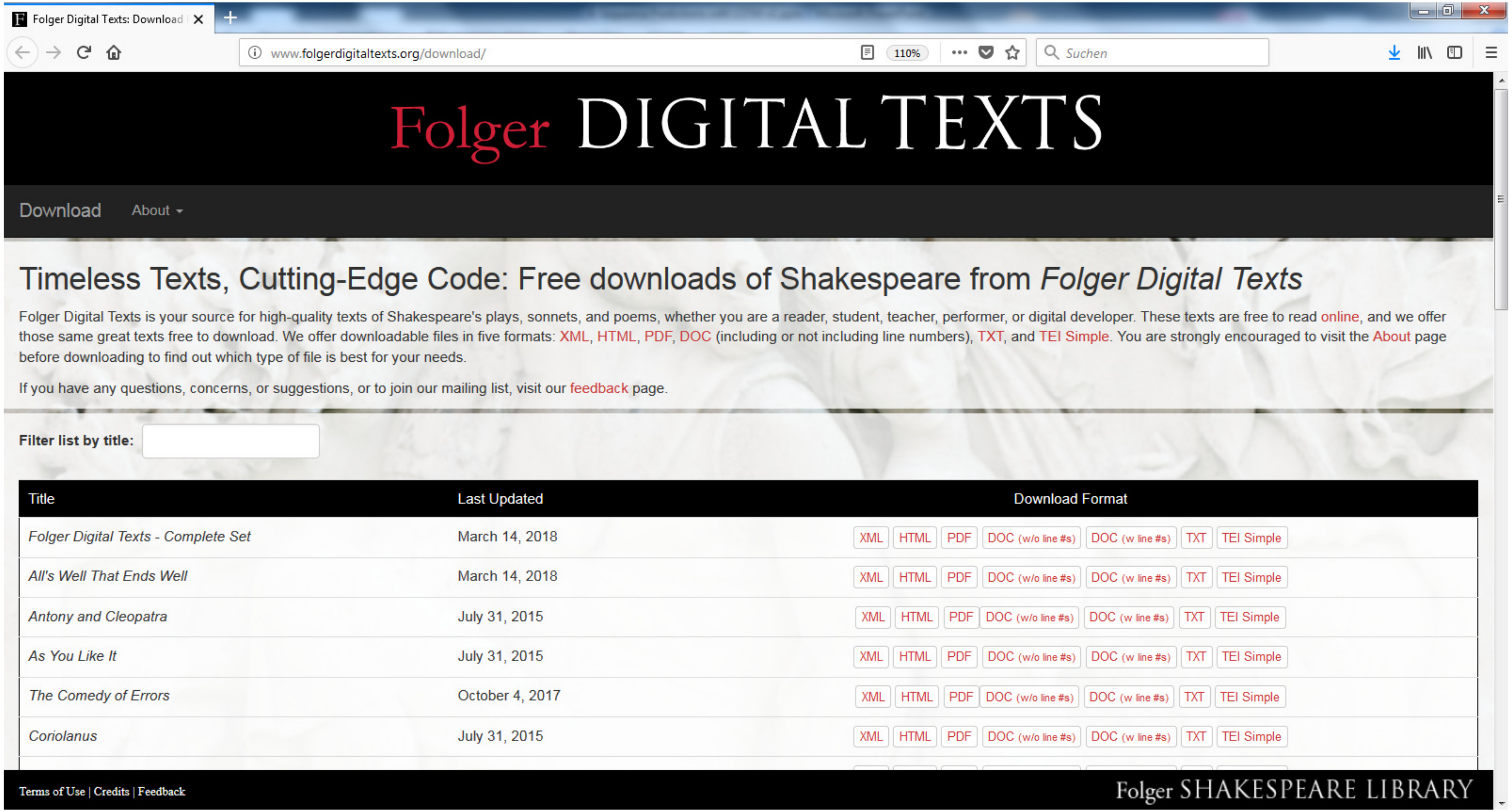




- Sequence values that are separated by a significant number of words (i.e. deep RNN) leads to the vanishing gradient problem (cf. day one)
- Reasoning is that small gradients or weights with values than 1 are multiplied many times through the multiple time steps, i.e. gradients shrink asymptotically to zero
- Effect is that weights of those earlier layers are not changed significantly and the network will not learn long-term dependencies

Exercises – RNN Example
Use Different number of Hidden Nodes, Epochs & Iterations

RNN Example – Data Repository

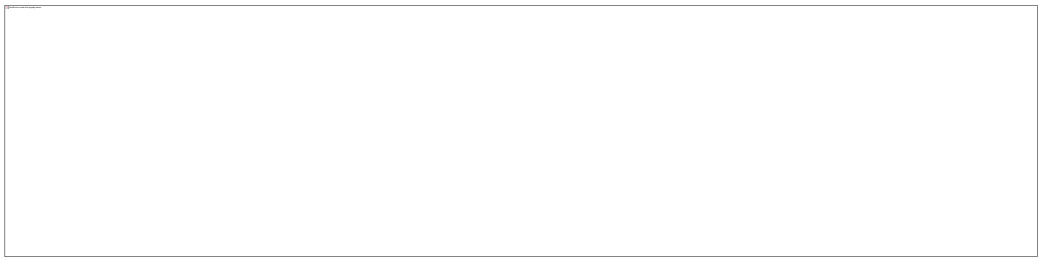


[7] Folger Digital Texts

RNN Example – Dataset & Application

- Unsupervised learning: simple sequence prediction example
  - Generating text by building a language model out of given text
  - Domain of Natural language processing (NLP)
  - Language models enable the prediction of the probability of a word in a given text given its previous words
  - Higher level applications: machine translation, spelling correction, etc.
- Data: Shakespeare text datasets
  - E.g. Shakespeare Macbeth (text)
  - http://www.folgerdigitaltexts.org/download/
  - Already downloaded and available on JURECA

[7] Folger Digital Texts



RNN Example – Dataset Exploration

- Use more/head/tail <textfile>
  - TBD: What challenges we see w.r.t. ‘clean datasets‘ & analysis?

```
-bash-4.2$ head Mac.txt
Macbeth
by William Shakespeare
Edited by Barbara A. Mowat and Paul Werstine
  with Michael Poston and Rebecca Niles
Folger Shakespeare Library
http://www.folgerdigitaltexts.org/?chapter=5&play=Mac
Created on Jul 31, 2015, from FDT version 0.9.2

Characters in the Play
======================
```

(metadata)

```
-bash-4.2$ tail Mac.txt
That fled the snares of watchful tyranny,
Producing forth the cruel ministers
Of this dead butcher and his fiend-like queen
(Who, as 'tis thought, by self and violent hands,
Took off her life)--this, and what needful else
That calls upon us, by the grace of grace,
We will perform in measure, time, and place.
So thanks to all at once and to each one,
Whom we invite to see us crowned at Scone.
[Flourish. All exit.]
```

(role commands)

RNN Example – Language Model Setup

- Typical approach
  - Create 'generative model' to predict the next word given previous words
  - Enables to generate text by sampling from the output probabilities
  - Build a 'word-based language model' → can be computational complex
- Simplified model for tutorial
  - Reasoning: simpler model and quicker training
  - Train a 'character based language model' on one text of Shakespeare
  - Take advantage of standard RNN cells
  - Predict (only) the next character given 10 previous characters
  - Use the trained language model to generate some text in the same style

(10 characters → prediction)

```
it turned -> i
 t turned i -> n
 turned in -> t
turned int -> o
urned into ->
rned into -> a
ned into a ->
ed into a -> p
d into a p -> i
 into a pi -> g
```

[8] Deep Learning with Keras

RNN Example – Keras Python Script – Preprocessing

```
from __future__ import print_function
from keras.layers import Dense, Activation
from keras.layers.recurrent import SimpleRNN
from keras.models import Sequential
import numpy as np
```

- Import necessary modules, e.g. SimpleRNN for a simple RNN cell, or Dense for a fully connected layer

```
# preprocessing of input text data
fin = open("/homea/hpclab/train001/data/macbeth/Mac.txt", 'rb')
lines = []
for line in fin:
  line = line.strip().lower()
  line = line.decode("ascii", "ignore")
  if len(line) == 0:
    continue
  lines.append(line)
fin.close()
text = " ".join(lines)
```

- Preprocessing of original files that e.g. contain line breaks, non-ASCII characters, capital characters; Result is variable text with 'cleaned text'

```
# lookup tables char vs index of chars
chars = set([c for c in text])
nb_chars = len(chars)
char2index = dict((c, i) for i, c in enumerate(chars))
index2char = dict((i, c) for i, c in enumerate(chars))
```

- Create lookup tables for characters per index & vice versa

- Character-level RNN: vocabulary is the set of characters that occur in the text → use index of character instead of a character itself

[8] Deep Learning with Keras

RNN Example – Keras Python Script – Input & Label Texts

```
# go through dataset, create input and label text
SEQLEN = 10
STEP = 1
input_chars = []
label_chars = []
for i in range(0, len(text) - SEQLEN, STEP):
  input_chars.append(text[i:i + SEQLEN])
  label_chars.append(text[i + SEQLEN])
```

```
# input and label text as vectors
X = np.zeros((len(input_chars), SEQLEN, nb_chars), dtype=np.bool)
y = np.zeros((len(input_chars), nb_chars), dtype=np.bool)
for i, input_char in enumerate(input_chars):
  for j, ch in enumerate(input_char):
    X[i, j, char2index[ch]] = 1
  y[i, char2index[label_chars[i]]] = 1
```

- **Task: Predict (only) the next character given 10 previous characters → SEQLEN = 10, STEP=1**

- **Moving step-wise through text by STEP=1 number of characters & extract span of text with size SEQLEN=10**

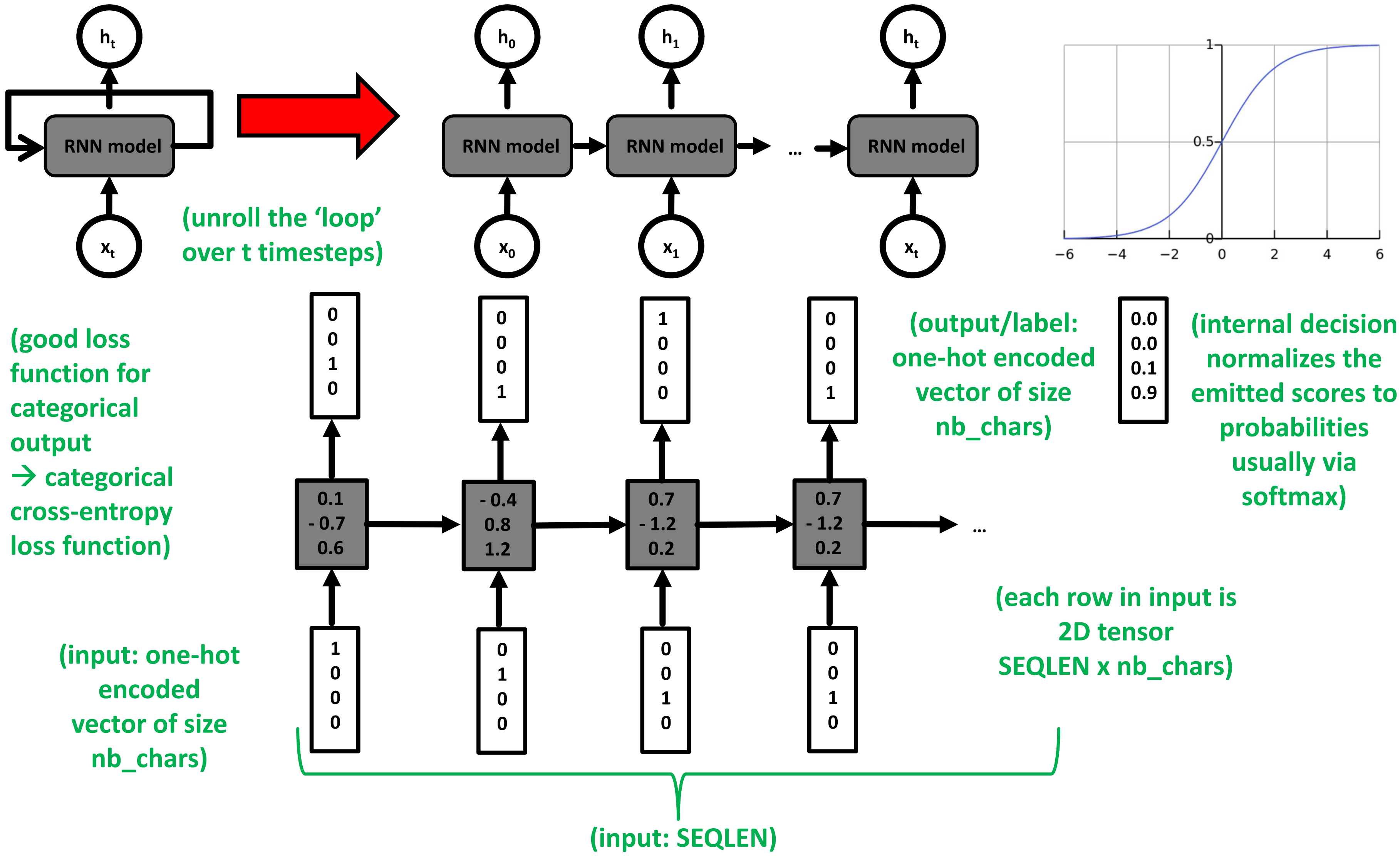
- **Each row of input to the RNN corresponds to one of the input texts**
- **SEQLEN characters input; vocabulary size = nb_chars (set of different characters in text) → one-hot encoded vector of size (nb_chars)**

(input_chars → label_chars)

```
it turned -> i
 t turned i -> n
 turned in -> t
turned int -> o
urned into ->
rned into -> a
ned into a ->
ed into a -> p
d into a p -> i
 into a pi -> g
```

[8] Deep Learning with Keras

RNN Example – Modelling & Decisions



RNN Example – Keras Python Script – Model & Parameter

```
# hyperparameter
HIDDEN_SIZE = 128
BATCH_SIZE = 128
NUM_ITERATIONS = 25
NUM_EPOCHS_PER_ITERATION = 1
NUM_PREDS_PER_EPOCH = 100
```

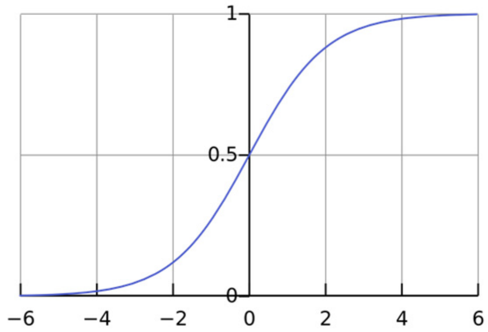
- Hyperparameter HIDDEN_SIZE=128 means output dimension of size 128 for ok text; parameter by experimentation

```
# RNN model
model = Sequential()
model.add(SimpleRNN(HIDDEN_SIZE, return_sequences=False,
  input_shape=(SEQLEN, nb_chars),
  unroll=True))
model.add(Dense(nb_chars))
model.add(Activation("softmax"))
model.compile(loss="categorical_crossentropy", optimizer="rmsprop")
```

- Sequential model adding first a SimpleRNN layer of size 128, return_sequences = False means single character as output/label not a ‘sequence of characters’, input tensor is SEQLEN x nb_chars; unroll = True - performance

- Adding a Dense layer of size nb_chars & activation function ‘softmax‘ (emits scores for each of the characters in vocabulary→ probabilities)
- Use optimizer ‘rmsprop‘ with ‘categorical_crossentropy‘ loss function

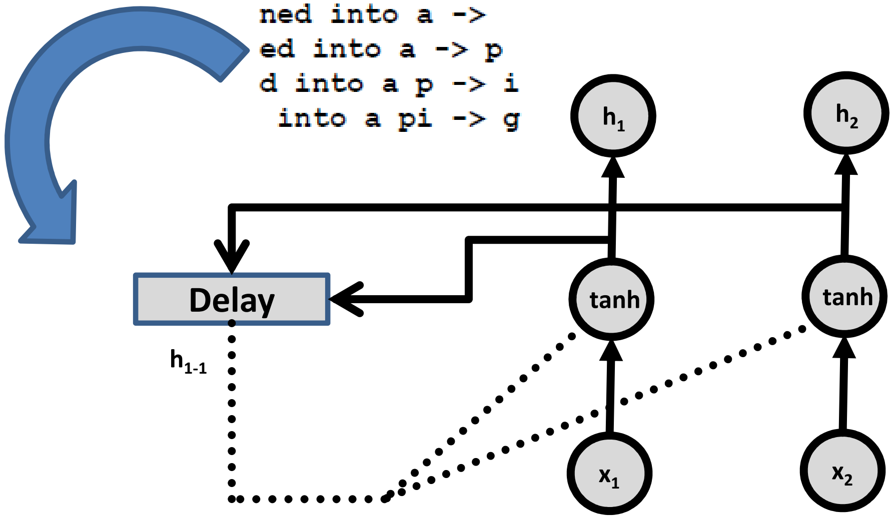
RNN Example – Keras Model & Activation Functions



(internal decision normalizes the emitted scores to probabilities usually via softmax)

```
it turned -> i
 t turned i -> n
 turned in -> t
turned int -> o
urned into ->
rned into -> a
ned into a ->
ed into a -> p
d into a p -> i
 into a pi -> g
```

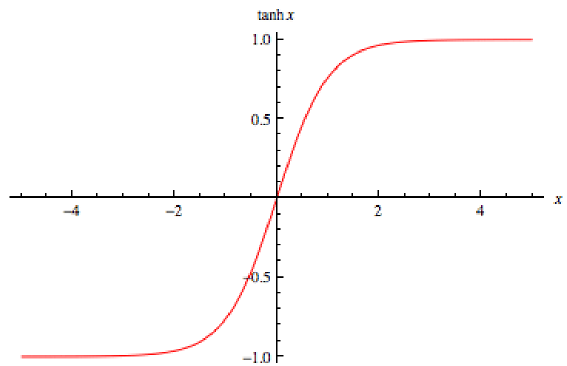
(iterations)



(Dense layer with 'number of characters' as nodes + 'softmax' activation function as output layer nodes)

(SimpleRNN layer with 128 hidden nodes with default hyperbolic tangent as activation function, i.e. values squashed between 1 and -1)

(input layer nodes)



RNN Example – Keras Python Script – Training Process

```
# training process
for iteration in range(NUM_ITERATIONS):
  print("=" * 50)
  print("Iteration #: %d" % (iteration))
  model.fit(X, y, batch_size=BATCH_SIZE, epochs=NUM_EPOCHS_PER_ITERATION)
  test_idx = np.random.randint(len(input_chars))
  test_chars = input_chars[test_idx]
  print("Generating from seed: %s" % (test_chars))
  print(test_chars, end="")
  for i in range(NUM_PREDS_PER_EPOCH):
    Xtest = np.zeros((1, SEQLEN, nb_chars))
    for i, ch in enumerate(test_chars):
      Xtest[0, i, char2index[ch]] = 1
    pred = model.predict(Xtest, verbose=0)[0]
    ypred = index2char[np.argmax(pred)]
    print(ypred, end="")
    # move forward with test_chars + ypred
    test_chars = test_chars[1:] + ypred
print()
```

- **Train model for epochs = 1 since no labelled dataset and then testing; training for 25 iterations → NUM_ITERATIONS; aka training for 25 epochs/iterations**

- **Test: generate a character from model given a random input; dropping the first character from the input & append the predicted character from our previous run & generate another character (100 x)**

- Cf. supervised learning process (day one)
  - Labels existing (not in this unsupervised example)
  - Train model for fixed number of epochs
  - Evaluate model against test dataset

[8] Deep Learning with Keras

RNN Example – Copy Keras Script & Job Script

```
/homea/hpclab/train001/tools/rnn
[train001@jrl04 rnn]$ ls -al
total 32
drwxr-xr-x  2 train001 hpclab  512 Jun  7 06:43 .
drwxr-xr-x 10 train001 hpclab  512 Jun  7 05:31 ..
-rw-r--r--  1 train001 hpclab 2349 Jun  7 06:43 rnn-example.py
-rw-r--r--  1 train001 hpclab  361 Jun  7 05:30 rnn-example-submit-juron.sh
```

- Create directory ‘rnn‘
- cp /homea/hpclab/train001/tools/rnn/rnn-example.py ~/rnn

```
-rw-r--r--  1 train001 hpclab    453 Jun  7 06:48 submit_train_simple_rnn.sh
```

- cp /homea/hpclab/train001/scripts/submit_train_simple_rnn.sh ~/rnn

RNN Example – Submit Script

- Job submit script
  - Specify good name for the job
  - Allocate GPUs for deep learning job
  - Specify job queue
  - Restore module environment with all dependencies
  - Use python with rnn-example.py script
- Use sbatch
  - Use jobscript

```
#!/bin/bash -x
#SBATCH--nodes=1
#SBATCH--ntasks=1
#SBATCH--output=rnn_out.%j
#SBATCH--error=rnn_err.%j
#SBATCH--time=01:00:00
#SBATCH--mail-user=m.riedel@fz-juelich.de
#SBATCH--mail-type=ALL
#SBATCH--job-name=simple-RNN

#SBATCH--partition=gpus
#SBATCH--gres=gpu:1

#SBATCH--reservation=deep_learning

### location executable
KERASSCRIPT=/homea/hpclab/train001/tools/rnn/rnn-example.py

module restore dl_tutorial

### submit
python $KERASSCRIPT
```

RNN Example – Output Interpretation

- Challenge: unsupervised learning problem
  - Check output with 'more out.txt'
  - Idea: string gives us an indication of the quality of the model
  - More epochs/iterations → better quality of the model

```
 97920/101872 [===========================>..] - ETA: 0s - loss: 2.5135
 99584/101872 [============================>.] - ETA: 0s - loss: 2.5090
101248/101872 [============================>.] - ETA: 0s - loss: 2.5046
101872/101872 [==============================] - 12s 117us/step - loss: 2.5029
Generating from seed:  but this
 but this the the the the the the the the the the the the the the the the the the
============
Iteration #: 1
Epoch 1/1
```

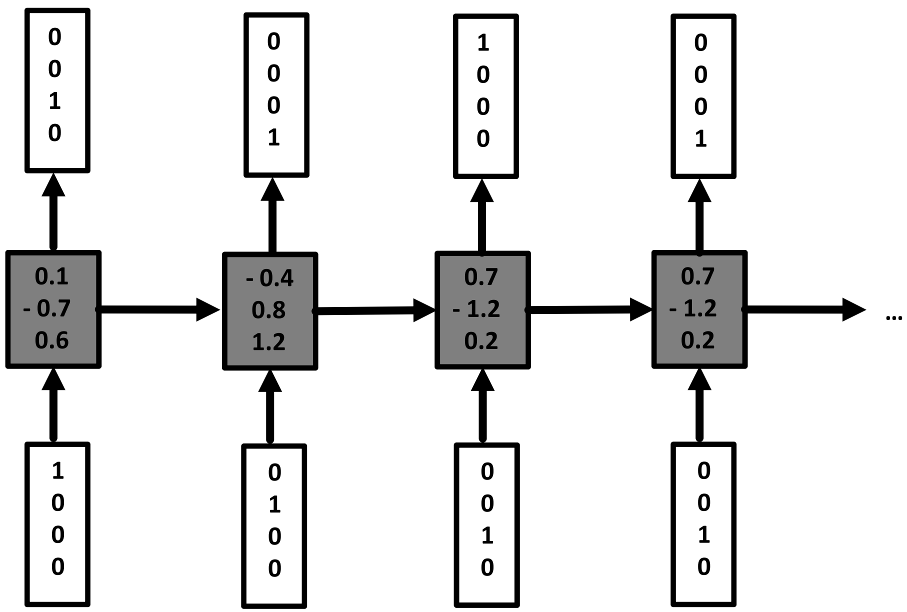
(learned well to spell compared to first iteration but no coherent thoughts → still interesting since no word concept)

```
 99968/101872 [============================>.] - ETA: 0s - loss: 1.5870
101632/101872 [============================>.] - ETA: 0s - loss: 1.5873
101872/101872 [==============================] - 3s 30us/step - loss: 1.5871
Generating from seed: eeks when
eeks when the did the pronor me the cantant in the with the dines and the servant he childred macbeth
============
Iteration #: 23
Epoch 1/1
```

RNN Topologies – Many-to-many (1)

- **RNN topologies express RNN capabilities to be arranged in many ways to solve specific problems**
- **Recall: RNNs combine the input data vector with the previous state vector to produce new states**
- **Common RNN topologies for sequences are driven by application problems but can be categorized roughly as follows: (a) many-to-many (1); (b) many-to-many (2); (c) one-to-many; (d) many-to-one**

- (a) many-to-many (1)
  - All input sequences are of the same length
  - Output is produced at each time step
- Example
  - RNN-Example above: Predicting next character



RNN Topologies – Many-to-many (2)

- (b) many-to-many (2)
  - Output / input data:
    Sequence-to-sequence network
- Example: machine translation network
  - Input: sequence of English words
  - Output: sequence of translated Spanish sentence
- Example: Part-of-Speech (POS) tagging
  - Input: words in a sentence
  - Output: corresponding POS tags

Grammar as a Foreign Language

Oriol Vinyals* Google vinyals@google.com

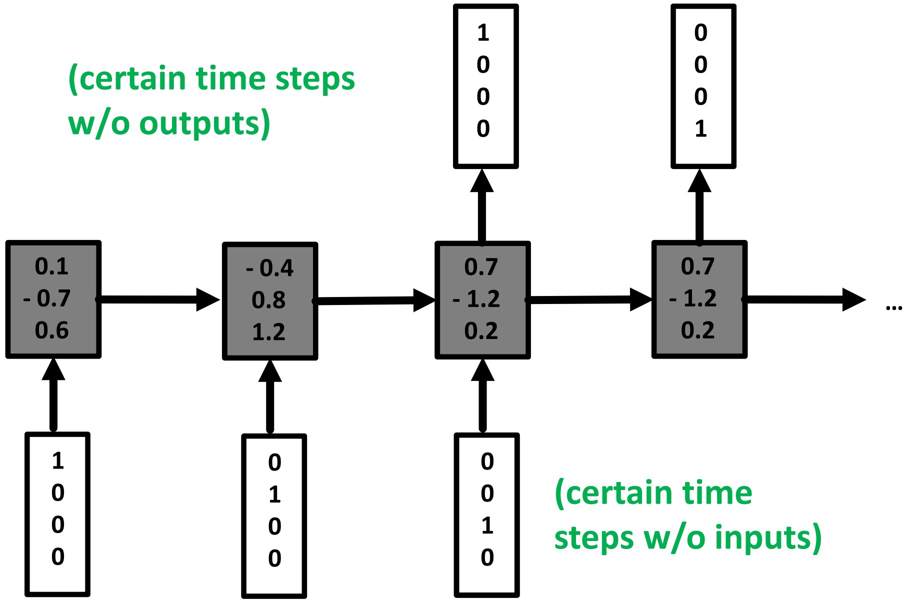
Lukasz Kaiser* Google lukaszkaiser@google.com

Terry Koo Google terrykoo@google.com

Slav Petrov Google slav@google.com

Ilya Sutskever Google ilyasu@google.com

Geoffrey Hinton Google geoffhinton@google.com



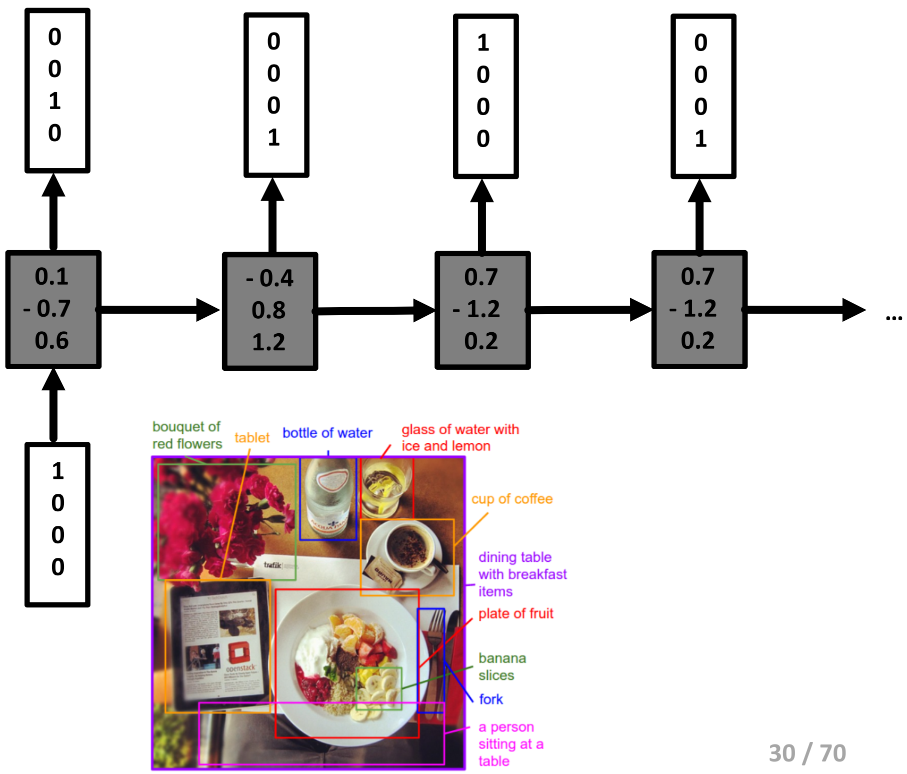
[9] O. Vinyals et al., 'Grammar as a Foreign Language'

RNN Topologies – One-to-many

- (c) one-to-many
  - E.g. different type of inputs combined with different types of outputs in a network
- Example: Image captioning network
  - Input: image
  - Output: sequence of words describing the image

Deep Visual-Semantic Alignments for Generating Image Descriptions

Andrej Karpathy Li Fei-Fei
Department of Computer Science, Stanford University
{karpathy,feifeili}@cs.stanford.edu

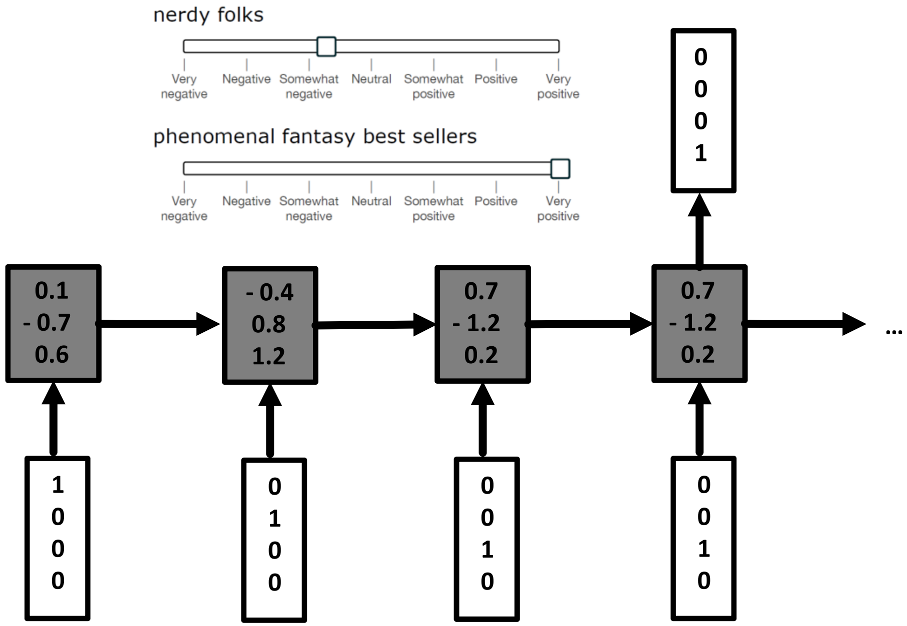


[10] A. Karpathy & F. Li, 'Deep Visual-Semantic Alignments for Generating Image Descriptions'

RNN Topologies – Many-to-one

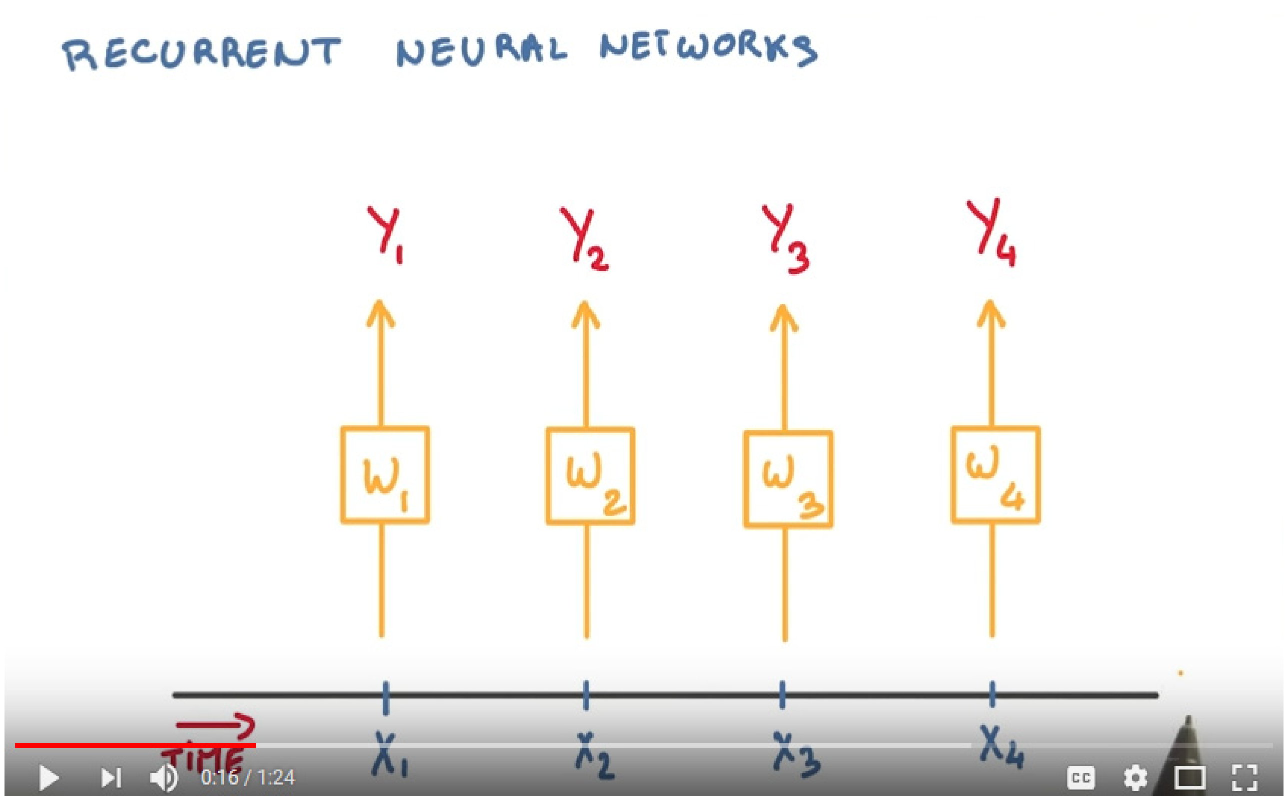
- (d) many-to-one
  - Summarize or judge a sequence of words & texts to a specific outcome
  - Often binary outcomes (good/negative)
- Example: Sentiment analysis of sentences
  - Input: Sequence of words (e.g. ratings, reviews, etc.)
  - Output: Positive/negative sentiment about input

[11] R. Socher et al., 'Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank'



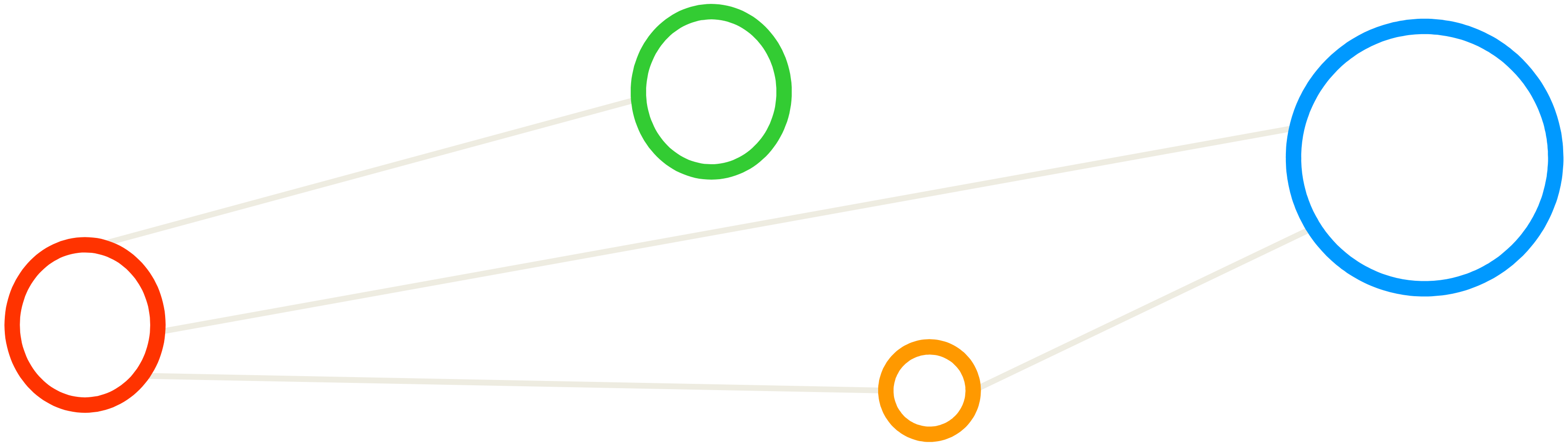
Exercises – RNN Example – Revisit Group Outputs

[Video] RNN Summary



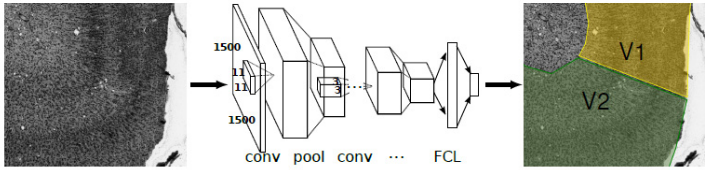
[12] RNNs, YouTube

Long Short-Term Memory



Deep Learning Architectures

- Deep Neural Network (DNN)
  - 'Shallow ANN' approach with many hidden layers between input/output
- Convolutional Neural Network (CNN, sometimes ConvNet)
  - Connectivity pattern between neurons is like animal visual cortex



- Deep Belief Network (DBN)
  - Composed of mult iple layers of variables; only connections between layers
- Recurrent Neural Network (RNN) → Long Short-Term Memory
  - RNN with state and temporal behaviour; LSTM adds 'strong memory'

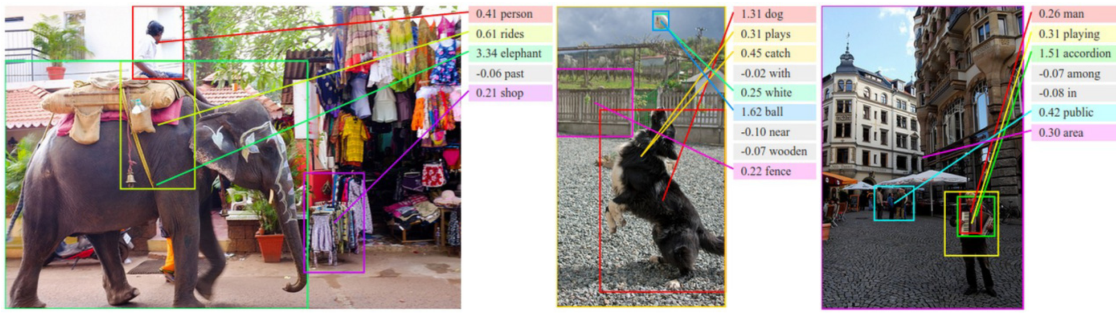
- **Deep Learning architectures can be classified into Deep Neural Networks, Convolutional Neural Networks, Deep Belief Networks, and Recurrent Neural Networks all with unique characteristica**
- **Deep Learning needs 'big data' to work well & for high accuracy – works not well on sparse data**

Different Useful LSTM Models

- Standard LSTM
  - Memory cells with single LSTM layer; used in simple network structures
- Stacked LSTM
  - LSTM layers are stacked one on top of another; creating deep networks
- CNN LSTM
  - CNNs to learn features (e.g. images); LSTM for image sequences
- Encoder-Decoder LSTM
  - One LSTM network → encode input; one LSTM network → decode output
- Bidirectional LSTM
  - Input sequences are presented and learned both forward & backwards
- Generative LSTM
  - LSTMs learn the inherent structure relationship in input sequences; then generate new plausible sequences

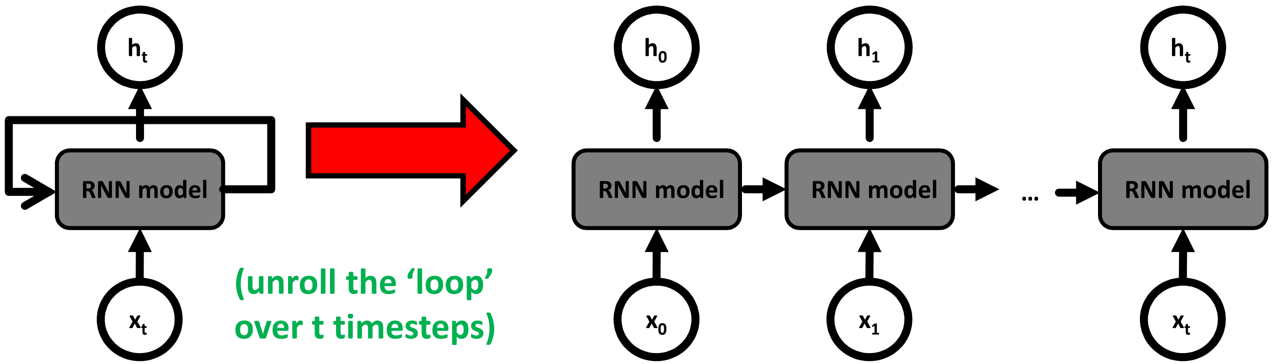
Long Short-Term Memory (LSTM) Models

- Specific type of Recurrent Neural Network (RNN)
  - Different to techniques like standard Artificial Neural Networks (ANNs) or Convolutional Neural Networks (CNNs)
  - Solving certain limits of ANNs through RNNs design
  - RNNs offer short-term memory – LSTMs add 'long-term' capabilities
  - Idea: improved performance through 'more memory' (cp. HPC?!)
- Designed specifically for sequence prediction problems
  - World-class results in complex problem domains & applications
  - E.g. language translation, automatic image captioning, text generation



[10] A. Karpathy & F. Li, 'Deep Visual-Semantic Alignments for Generating Image Descriptions'

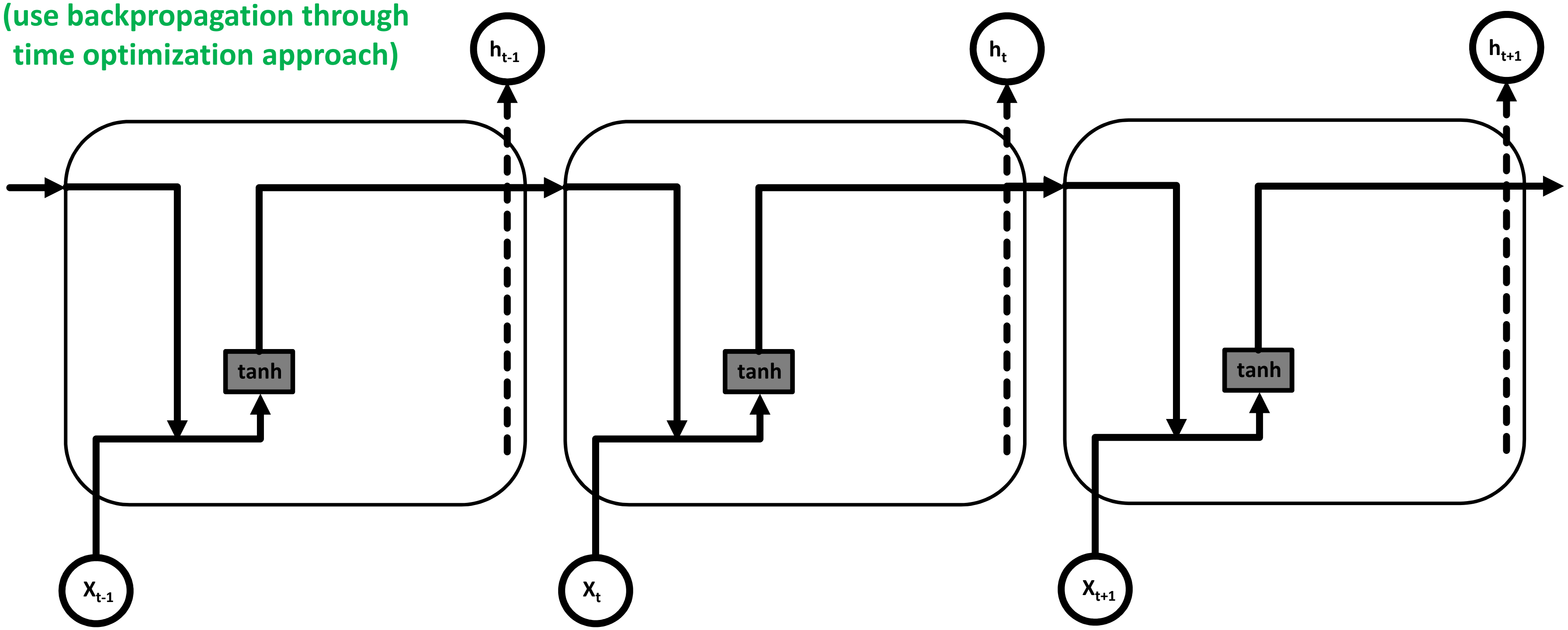
Unrolled RNN – Revisited



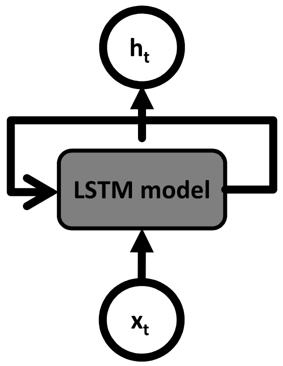
(unroll the 'loop' over t timesteps)

- A RNN can be viewed as multiple copies of the same network, each passing a message to a successor – this gets clear when 'unrolling the RNN loop'

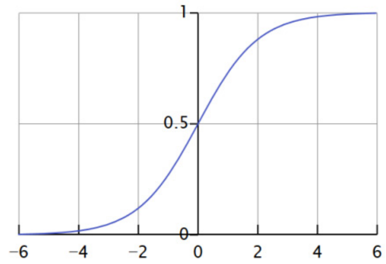
(use backpropagation through time optimization approach)



Long Short Term Memory (LSTM) Model

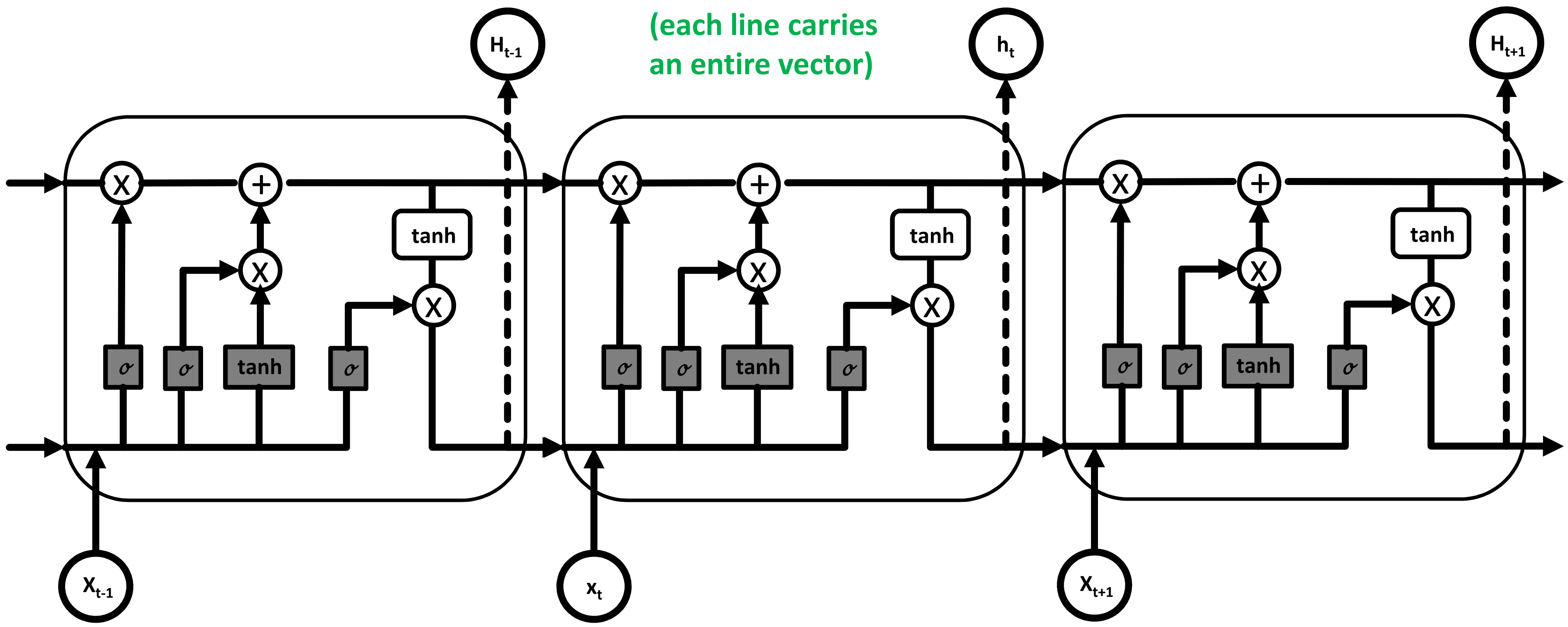


- Long Short Term Memory (LSTM) networks are a special kind of Recurrent Neural Networks (RNNs)
- LSTMs learn long-term dependencies in data by remembering information for long periods of time
- The LSTM chain structure consists of four neural network layers interacting in a specific way



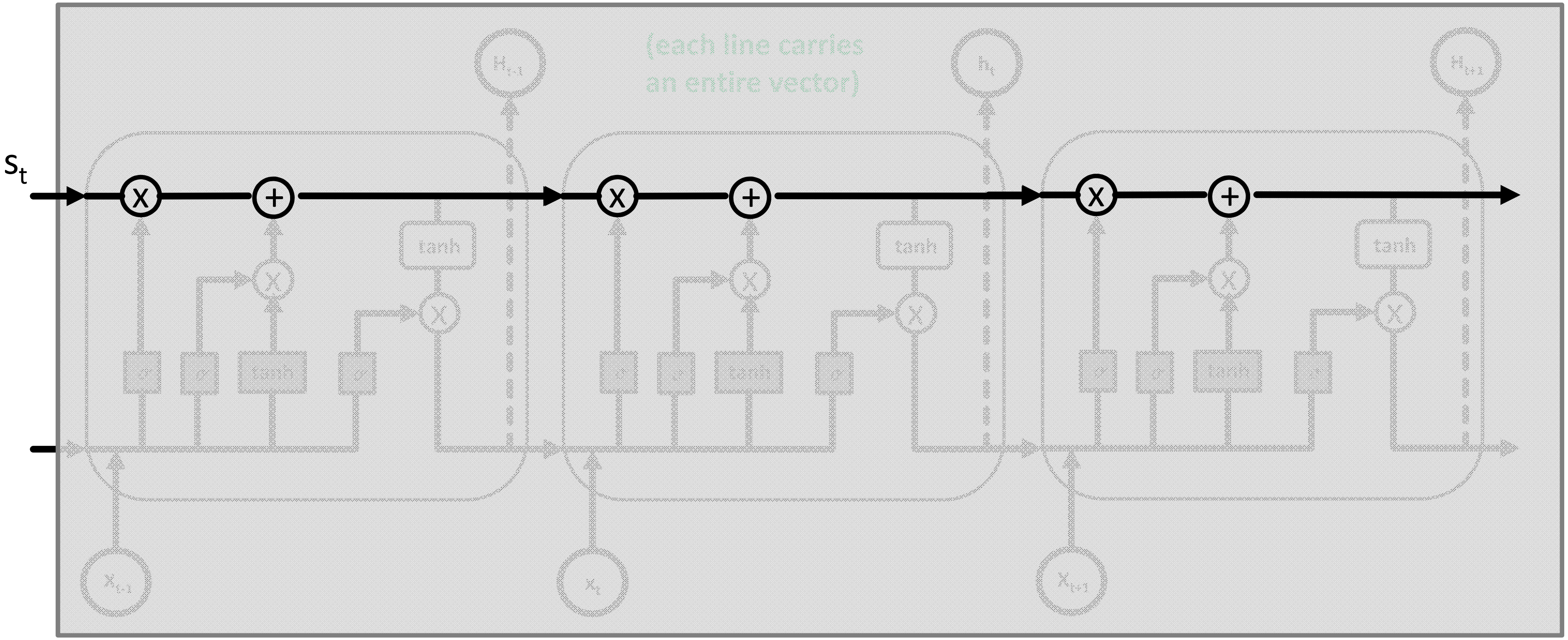
(uses sigmoid σ)

(each line carries an entire vector)



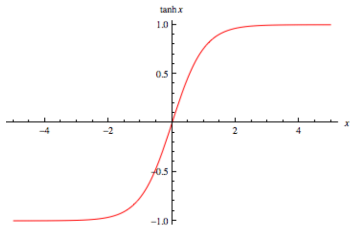
LSTM Model – Memory Cell & Cell State

- LSTM introduce a 'memory cell' structure into the underlying basic RNN architecture using four key elements: an input gate, a neuron with self-current connection, a forget gate, and an output gate
- The data in the LSTM memory cell flows straight down the chain with some linear interactions (x,+)
- The cell state s_t can be different at each of the LSTM model steps & modified with gate structures
- Linear interactions of the cell state are pointwise multiplication (x) and pointwise addition (+)
- In order to protect and control the cell state s_t three different types of gates exist in the structure

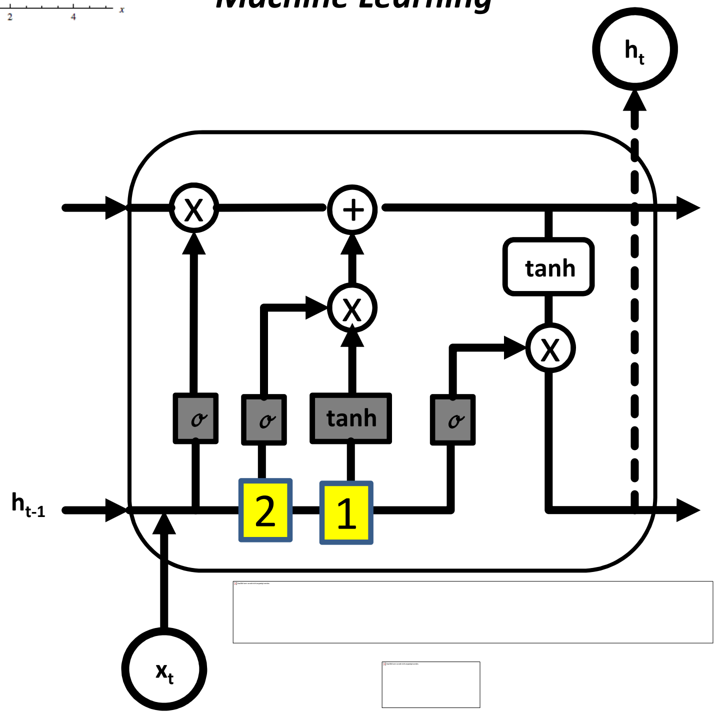


Computing of LSTM Cell – Step 1-2

1. New x_t input together with the output from cell h_{ht-1} are squashed via a tanh layer
   - Outputs between -1 and 1

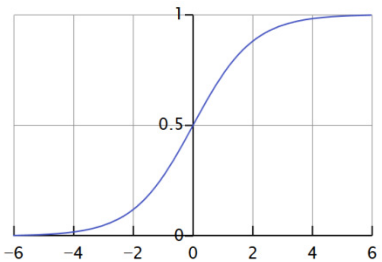


[14] Adventures in Machine Learning



2. New x_t input together with the output from cell h_{ht-1} is passed through the 'input gate'
   - Layer of sigmoid activated nodes whose output is multiplied by squashed input

$$i = \sigma(b^i + x_t U^i + h_{t-1} V^i)$$



(uses sigmoid σ)

(gate sigmoid σ can act to 'switch off' any elments of the input vector that are not required)

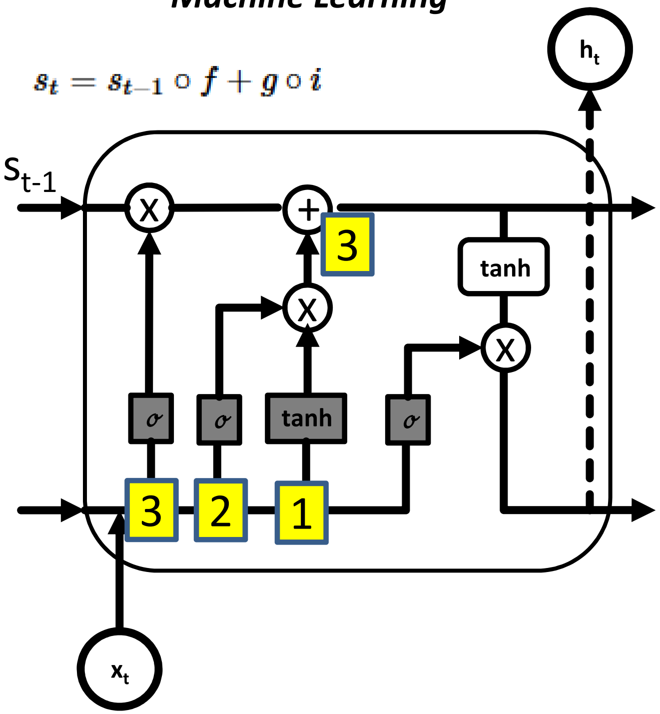
(sigmoid function outputs values between 0 and 1, weights connecting the input to these nodes can be trained to output values close to zero to 'switch off' certain input values – or outputs close to 1 to 'pass through')

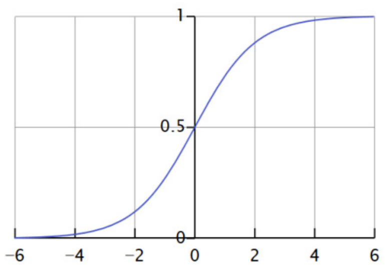
Computing of LSTM Cell – Step 3

3. Internal state / forget gate $f = \sigma(b^f + x_t U^f + h_{t-1} V^f)$
   - LSTM cells have internal cell state s_t
   - 'Delay' – lagged one time step: s_{t-1}
   - Added to the input data to create an effective 'layer of recurrence'
   - Addition instead of 'usual' multiplication reduces risk of vanishing gradients
   - The connection to cell state is carefully controlled by a forget gate with sigmoid (works like the input gate)

[14] Adventures in Machine Learning

$s_t = s_{t-1} \circ f + g \circ i$



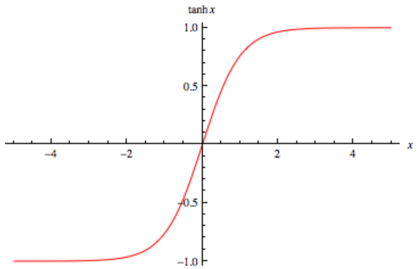


(uses sigmoid σ)

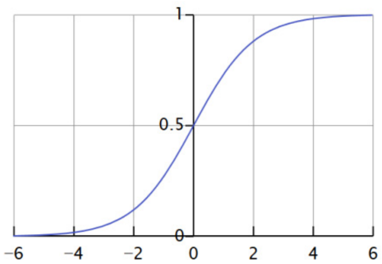
(gate sigmoid σ can act to 'switch off' any elments of the cell state to steer what variables should be remembered or forgotten)

Computing of LSTM Cell – Step 4

4. Output layer & output gate $o = \sigma(b^o + x_t U^o + h_{t-1} V^o)$
   - Output layer with tanh squashing function



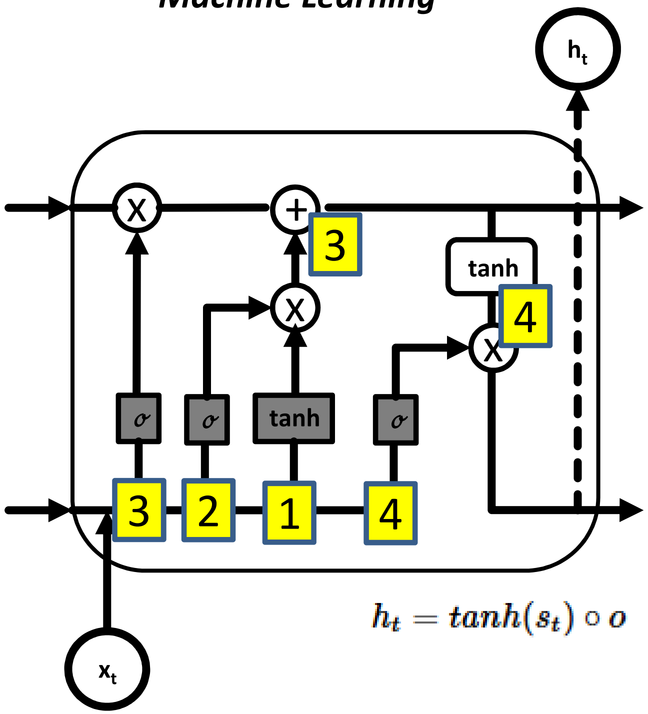
   - Output is controlled via output gate with sigmoid activation function



(uses sigmoid σ)

(gate sigmoid σ can learn to determine which values are allowed as an output from the cell)

[14] Adventures in Machine Learning

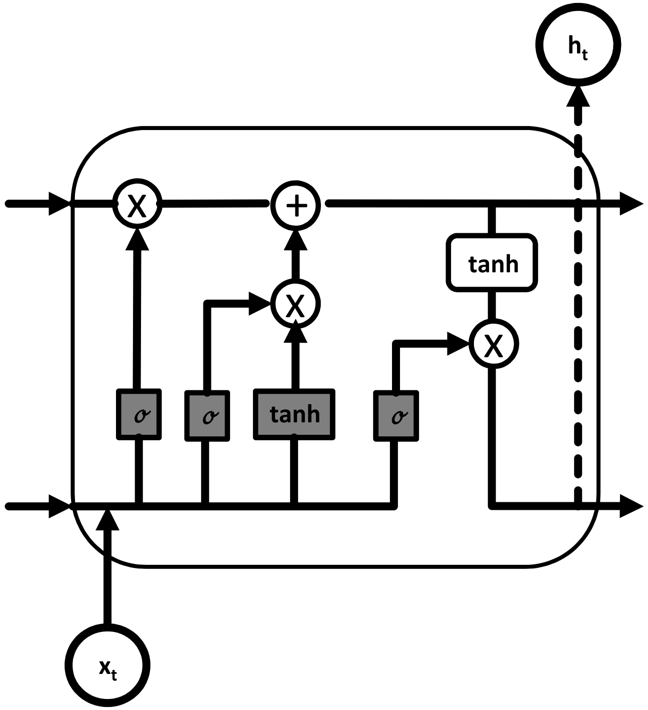


$h_t = tanh(s_t) \circ o$

Low-level Tools – Theano

- Theano is a low-level deep learning library implemented in Python with a focus on defining, optimizing, and evaluating mathematical expressions & multi-dimensional arrays
- The Theano tool supports the use of GPUs and CPUs via expressions in NumPy syntax
- Theano work with the high-level deep learning tool Keras in order to create models fast
- LSTM models are created using mathematical equations but there is no direct class for it

```
...
import numpy
import theano
from theano import config
import theano.tensor as tensor
...
def lstm_layer(tparams, state_below,
   options, prefix='lstm', mask=None):
...
i = tensor.nnet.sigmoid(_slice(preact, 0,
                        options['dim_proj']))
f = tensor.nnet.sigmoid(_slice(preact, 1,
                        options['dim_proj']))
o = tensor.nnet.sigmoid(_slice(preact, 2,
                        options['dim_proj']))
c = tensor.tanh(_slice(preact, 3,
                        options['dim_proj']))
```



[2] Theano Deep Learning Framework *[3] LSTM Networks for Sentiment Analysis*

Low-Level Tools – Tensorflow

- Tensorflow is an open source library for deep learning models using a flow graph approach
- Tensorflow nodes model mathematical operations and graph edges between the nodes are so-called tensors (also known as multi-dimensional arrays)
- The Tensorflow tool supports the use of CPUs and GPUs (much more faster than CPUs)
- Tensorflow work with the high-level deep learning tool Keras in order to create models fast
- LSTM models are created using tensors & graphs and there are LSTM package contributions

[4] Tensorflow Deep Learning Framework

TensorFlow

```
...
lstm = rnn_cell.BasicLSTMCell(lstm_size, state_is_tuple=False)
...
stacked_lstm = rnn_cell.MultiRNNCell([lstm] * number_of_layers,
    state_is_tuple=False)
...
initial_state = state = stacked_lstm.zero_state(batch_size, tf.float32)

for i in range(num_steps):
    # The value of state is updated
    # after processing each batch of words.
    output, state = stacked_lstm(words[:, i], state)

    # The rest of the code.
    # ...

final_state = s
```

- The class BasicLSTMCell() offers a simple LSTM Cell implementation in Tensorflow

High-level Tools – Keras

- Keras is a high-level deep learning library implemented in Python that works on top of existing other rather low-level deep learning frameworks like Tensorflow, CNTK, or Theano
- The key idea behind the Keras tool is to enable faster experimentation with deep networks
- Created deep learning models run seamlessly on CPU and GPU via low-level frameworks

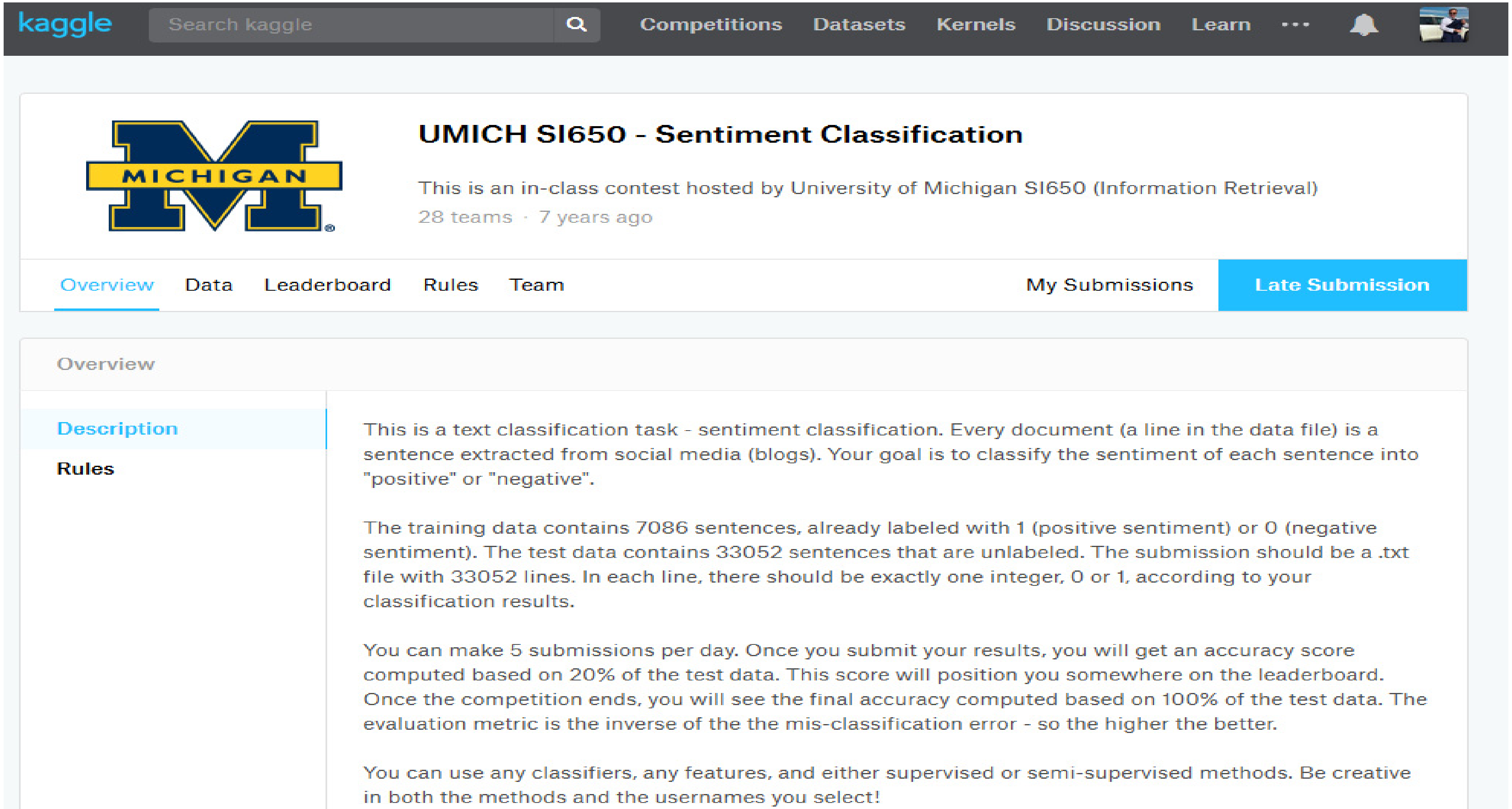
```
keras.layers.LSTM(  units,
                    activation='tanh',
                    recurrent_activation='hard_sigmoid',
                    use_bias=True,
                    kernel_initializer='glorot_uniform',
                    recurrent_initializer='orthogonal',
                    bias_initializer='zeros',
                    unit_forget_bias=True,
                    kernel_regularizer=None,
                    recurrent_regularizer=None,
                    bias_regularizer=None,
                    activity_regularizer=None,
                    kernel_constraint=None,
                    recurrent_constraint=None,
                    bias_constraint=None,
                    dropout=0.0, ...)
```

- Tool Keras supports the LSTM model via keras.layers.LSTM() that offers a wide variety of configuration options

K Keras *[1] Keras Python Deep Learning Library*

Exercises – LSTM Example
Use Different number of Hidden Nodes, Epochs & Iterations

LSTM Example – Data Repository



[13] Kaggle, UMICH SI650 – Sentiment Classification Data

LSTM Example – Dataset & Application

- Sentiment analysis (many-to-one RNN topology)
  - Input: sentence as sequence of words (i.e. movie ratings texts)
  - Output: Sentiment value (positive/negative movie rating)
  - Application was a former competition (i.e. Kaggle platform overall idea)
  - Goal: Create LSTM network that will learn to predict a correct sentiment
- Small dataset example for tutorial: training & test data available
  - Training samples: 7086 short sentences (labelled) [~440 KB]
  - Test samples: 33052 short sentences[~1.94 MB]
  - Format: label & tab seperated sentence
  - https://www.kaggle.com/c/si650winter11/data

Data Description

Training data: 7086 lines.
Format: 1|0 (tab) sentence

Test data: 33052 lines, each contains one sentence.

[13] Kaggle, UMICH SI650 – Sentiment Classification Data

LSTM Example – Dataset Exploration

- Create directory lstm
- Copy data
  - cp /homea/hpclab/train001/data/sentiments/* ~/lstm

```
/homea/hpclab/train001/data/sentiments
[train001@jrl04 sentiments]$ ls -al
total 2912
drwxr-xr-x  2 train001 hpclab     512 Jun  7 06:33 .
drwxr-xr-x 12 train001 hpclab     512 Jun  7 05:55 ..
-rw-r--r--  1 train001 hpclab 2033345 Jun  7 06:44 testdata.txt
-rw-r--r--  1 train001 hpclab  447540 Jun  7 06:16 training-original.txt
-rw-r--r--  1 train001 hpclab  447540 Jun  7 06:10 training.txt
```

```
-bash-4.2$ head training.txt
1       The Da Vinci Code book is just awesome.
1       this was the first clive cussler i've ever read, but even books like Relic, and Da Vinci code were more plausible than this.
1       i liked the Da Vinci Code a lot.
1       i liked the Da Vinci Code a lot.
1       I liked the Da Vinci Code but it ultimatly didn't seem to hold it's own.
1       that's not even an exaggeration ) and at midnight we went to Wal-Mart to buy the Da Vinci Code, which is amazing of course.
1       I loved the Da Vinci Code, but now I want something better and different!..
1       i thought da vinci code was great, same with kite runner.
1       The Da Vinci Code is actually a good movie...
1       I thought the Da Vinci Code was a pretty good book.
```

(labelled training dataset)

```
-bash-4.2$ head testdata.txt
" I don't care what anyone says, I like Hillary Clinton.
have an awesome time at purdue!..
Yep, I'm still in London, which is pretty awesome: P Remind me to post the million and one pictures that I took when I get back to Markham!...
Have to say, I hate Paris Hilton's behavior but I do think she's kinda cute..
i will love the lakers.
I'm so glad I love Paris Hilton, too, or this would be excruciating.
considering most Geico commericals are stupid...
i liked MIT though, esp their little info book(
Before I left Missouri, I thought London was going to be so good and cool and fun and a really great experience and I was really excited.
I still like Tom Cruise.
```

(testing dataset)

LSTM Example – Keras Python Script – Preprocessing

```
from keras.layers.core import Activation, Dense, Dropout, SpatialDropout1D
from keras.layers.embeddings import Embedding
from keras.layers.recurrent import LSTM
from keras.models import Sequential
from keras.preprocessing import sequence
from sklearn.model_selection import train_test_split
import collections
import matplotlib.pyplot as plt
import nltk
import numpy as np
import os

# obtain punkt if not there already
nltk.download('punkt')

# define a data directory
DATA_DIR = "/homea/hpclab/train001/data/sentiments"
```

- Import necessary modules, e.g. LSTM for a simple LSTM cell, or Dense for a fully connected layer
- Import good sklearn model selection tools
- Import numpy for as helper tool

- Location for labeled training data and testset data

```
/homea/hpclab/train001/data/sentiments
[train001@jrl04 sentiments]$ ls -al
total 2912
drwxr-xr-x  2 train001 hpclab     512 Jun  7 06:33 .
drwxr-xr-x 12 train001 hpclab     512 Jun  7 05:55 ..
-rw-r--r--  1 train001 hpclab 2033345 Jun  7 06:44 testdata.txt
-rw-r--r--  1 train001 hpclab  447540 Jun  7 06:16 training-original.txt
-rw-r--r--  1 train001 hpclab  447540 Jun  7 06:10 training.txt
```

- Natural Language Toolkit (NLTK) is for building Python programs working on human language datasets (punkt is tokenizer)

[8] Deep Learning with Keras

LSTM Example – Keras Python Script – Vocabulary Setup

```
# exploratory analysis
maxlen = 0
word_freqs = collections.Counter()
num_recs = 0
ftrain = open(os.path.join(DATA_DIR, "training.txt"), 'rb')
for line in ftrain:
   label, sentence = line.strip().split('\t')
   words = nltk.word_tokenize(sentence.decode("ascii", "ignore").lower())
   if len(words) > maxlen:
      maxlen = len(words)
   for word in words:
      word_freqs[word] += 1
   num_recs += 1
ftrain.close()
```

- Perform exploratory analysis in order to find out the number of unique words in the whole corpus & how many words are roughly in each sentence

```
# vocabulary setup
MAX_FEATURES = 2000
MAX_SENTENCE_LENGTH = 40

# vocabulary and indices
vocab_size = min(MAX_FEATURES, len(word_freqs)) + 2
word2index = {x[0]: i+2 for i, x in
enumerate(word_freqs.most_common(MAX_FEATURES))}
word2index["PAD"] = 0
word2index["UNK"] = 1
index2word = {v:k for k, v in word2index.items()}
```

- Exploration reveals maxlen: 42 & len(word_freqs): 2313
- Number of words in sentence (maxlen) enables a fixed sequence length & PAD = 0; truncate long ones

- Creating indices index2word and vice versa
- Out of vocabulary means UNK (unknown)

[8] Deep Learning with Keras

LSTM Example – Keras Python Script – Indices & Padding

```
# input sentence -> word index sequences
X = np.empty((num_recs, ), dtype=list)
y = np.zeros((num_recs, ))
i = 0
ftrain = open(os.path.join(DATA_DIR, "training.txt"), 'rb')
for line in ftrain:
  label, sentence = line.strip().split('\t')
  words = nltk.word_tokenize(sentence.decode("ascii", "ignore").lower())
  seqs = []
  for word in words:
    if word2index.has_key(word):
      seqs.append(word2index[word])
    else:
      seqs.append(word2index["UNK"])
  X[i] = seqs
  y[i] = int(label)
  i += 1
ftrain.close()
```

- **Convert input sentences from the training data to word index sequences and add unknown ones as UNK in index**

```
# perform padding if needed
X = sequence.pad_sequences(X, maxlen=MAX_SENTENCE_LENGTH)
```

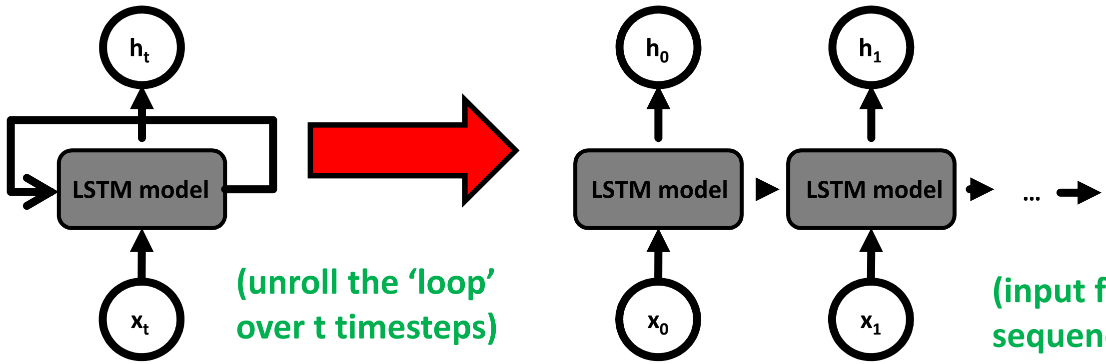
- **Perform padding to the maximum sentence length (40)**
- **Labels are binary (positive/negative sentiment) and do not need padding**

```
# split into training and testing
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y,
        test_size=0.2, random_state=42)
```

- **Split between training & testing set (ratio rule of thumb 80:20)**
- **There is another test set put aside for nicely checking out-of-sample**

[8] Deep Learning with Keras

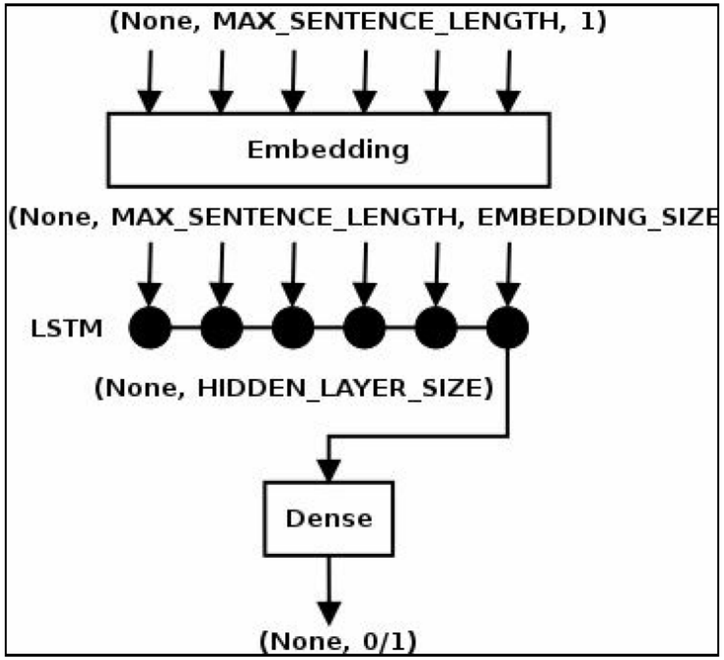
LSTM Example – Modelling & Decisions



(unroll the 'loop' over t timesteps)

(input for each row is a sequence of word indices – sequence length is given by MAX_SENTENCE_LENGTH)

(tensor layout: None X MAX_SENTENCE_LENGTH X 1)



(tensor dimensions: first is None →
indicate that the batch size is currently unknown,
i.e. number of records fed to the network →
defined in runtime using BATCH_SIZE parameter)

(tensor fed to embedding layer →
weights are initialized with small random values & learned
i.e. layer transforms the tensor to a shape of
None X MAX_SENTENCE_LENGTH X EMBEDDING_SIZE)

(output of LSTM is the tensor
None X HIDDEN_LAYER_SIZE, because last tensor can be defined
as return_sequences = False → we just need 0/1 output)

(Dense layer with Sigmoid activation function →
0 – negative review / 1 positive review)

modified from [8] Deep Learning with Keras

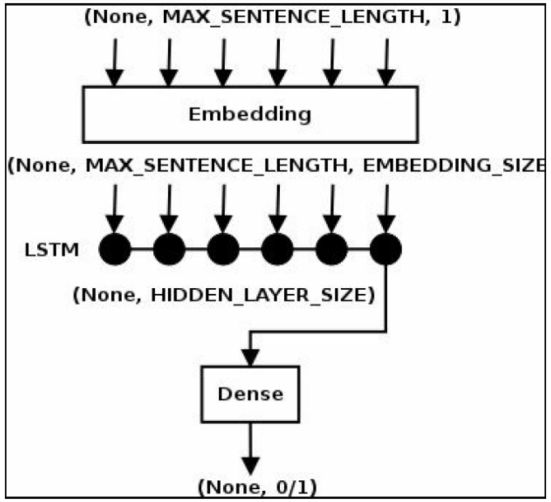
LSTM Example – Keras Python Script – Model & Parameter

```
#hyperparameter
EMBEDDING_SIZE = 128
HIDDEN_LAYER_SIZE = 64
BATCH_SIZE = 32
NUM_EPOCHS = 10
```

```
# LSTM model
model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_SIZE,
input_length=MAX_SENTENCE_LENGTH))
model.add(SpatialDropout1D(Dropout(0.2)))
model.add(LSTM(HIDDEN_LAYER_SIZE, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1))
model.add(Activation("sigmoid"))
model.compile(loss="binary_crossentropy", optimizer="adam",
  metrics=["accuracy"])
```

- Hyperparameters embedding=128; hidden layers=64; parameter by experimentation

- Create first embedding layer with input tensor None X maximum sentence length X 1
- Add regularizer SpatialDropout1D
- Add LSTM cell with hidden layer size 64 with regularizers dropout and recurrent_dropout
- Add Dense layer and Sigmoid activation



- All hyperparameters are tuned experimentally over many runs
- Compile model using binary cross-entropy loss function good for a binary model used here
- Use of Adam optimizer as good general purpose optimizer

[8] Deep Learning with Keras

LSTM Example – Keras Python Script – Train & Evaluate

```
# training
story = model.fit(Xtrain, ytrain, batch_size=BATCH_SIZE, epochs=NUM_EPOCHS,
validation_data=(Xtest, ytest))
```

```
# evaluation
score, acc = model.evaluate(Xtest, ytest, batch_size=BATCH_SIZE)
print("Test score: %.3f, accuracy: %.3f" % (score, acc))
for i in range(5):
  idx = np.random.randint(len(Xtest))
  xtest = Xtest[idx].reshape(1,40)
  ylabel = ytest[idx]
  ypred = model.predict(xtest)[0][0]
  sent = " ".join([index2word[x] for x in xtest[0].tolist() if x != 0])
  print("%.0ft  %dt  %s" % (ypred, ylabel, sent))
```

- **Train the LSTM network for 10 epochs (NUM_EPOCHS) & with batch size 32**
- **Perform validation at each epoch using test data**

- **Evaluate model against the full test set showing score and accuracy**
- **Show the LSTM prediction with pick of a few random sentences from the test set (predicted label, label & actual sentence**

- Cf. supervised learning process (day one)
  - Labels existing (not in this unsupervised example)
  - Train model for fixed number of epochs
  - Evaluate model against test dataset (splitted training)
- TBD (home work): Use model for prediction of the real 'test-data' (not splitted training)
  - Note: real 'test-data' has no labels, aka unseen data

RNN Example – Copy Keras Script & Job Script

```
[train001@jrl04 lstm]$ pwd
/homea/hpclab/train001/tools/lstm
[train001@jrl04 lstm]$ ls -al
total 32
drwxr-xr-x  2 train001 hpclab  512 Jun  7 07:40 .
drwxr-xr-x 10 train001 hpclab  512 Jun  7 05:31 ..
-rw-r--r--  1 train001 hpclab 2993 Jun  7 05:31 lstm-example.py
-rw-r--r--  1 train001 hpclab  366 Jun  7 05:31 lstm-example-submit-juron.sh
```

- Create directory ‘lstm‘
- cp /homea/hpclab/train001/tools/lstm/lstm-example.py ~/lstm

```
-rw-r--r--  1 train001 hpclab    454 Jun  7 07:34 submit_train_simple_lstm.sh
```

- cp /homea/hpclab/train001/scripts/submit_train_simple_lstm.sh ~/lstm

LSTM Example – Submit Script (JURECA)

- Job submit script
  - Specify good name for the job
  - Allocate GPUs for deep learning job
  - Specify job queue
  - Restore module environment with all dependencies
  - Use python with lstm-example.py script
- Use sbatch
  - Use job script

```
#!/bin/bash -x
#SBATCH--nodes=1
#SBATCH--ntasks=1
#SBATCH--output=lstm_out.%j
#SBATCH--error=lstm_err.%j
#SBATCH--time=01:00:00
#SBATCH--mail-user=m.riedel@fz-juelich.de
#SBATCH--mail-type=ALL
#SBATCH--job-name=simple-LSTM

#SBATCH--partition=gpus
#SBATCH--gres=gpu:1

#SBATCH--reservation=deep_learning

### location executable
KERASSCRIPT=/homea/hpclab/train001/tools/lstm/lstm-example.py

module restore dl_tutorial

### submit
python $KERASSCRIPT
```

LSTM Example – Output Interpretation

- Supervised learning problem
  - Check output with ‘more out.txt‘
  - Idea: predicted sentiment should be closed to sentiment labels
  - More epochs/iterations → better quality of the model

(learned well compared to first iteration → one can observe loss decrease and increase in accuracy over multiple epochs)

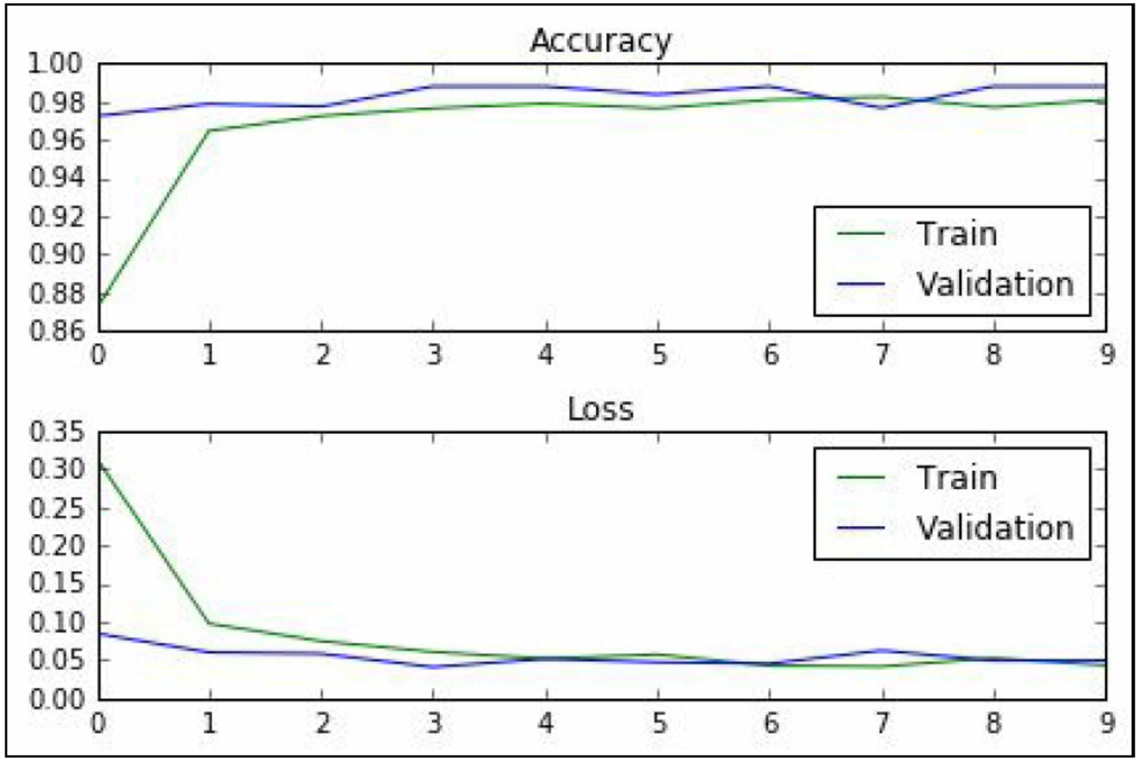
```
Train on 5668 samples, validate on 1418 samples
Epoch 1/10

  32/5668 [..............................] - ETA: 35:08 - loss: 0.6938 - acc: 0.4688
  64/5668 [..............................] - ETA: 17:36 - loss: 0.6927 - acc: 0.5312
  96/5668 [..............................] - ETA: 11:45 - loss: 0.6911 - acc: 0.5625
```

```
5664/5668 [============================>.] - ETA: 0s - loss: 0.0015 - acc: 0.9995
5668/5668 [==============================] - 15s 3ms/step - loss: 0.0015 - acc: 0.9995 - val_loss: 0.0845 - val_acc: 0.9718
Epoch 10/10

  32/5668 [..............................] - ETA: 13s - loss: 0.0697 - acc: 0.9688
  64/5668 [..............................] - ETA: 13s - loss: 0.0353 - acc: 0.9844
  96/5668 [..............................] - ETA: 13s - loss: 0.0240 - acc: 0.9896
```

```
Test score: 0.072, accuracy: 0.980
1t  1t  the people who are worth it know how much i love the da vinci code .
1t  1t  anyway , thats why i love `` brokeback mountain .
0t  0t  the da vinci code sucked .
0t  0t  this quiz sucks and harry potter sucks ok bye..
1t  1t  because i would like to make friends who like the same things i like ,
```

LSTM Example – Model Evaluation

- Selected plots (e.g. for papers)
  - E.g. matplotlib & pyplot can be used to create simple graphs



[8] Deep Learning with Keras

Different Useful LSTM Models – Many other Applications

- Standard LSTM
  - Memory cells with single LSTM layer; used in simple network structures
- Stacked LSTM
  - LSTM layers are stacked one on top of another; creating deep networks
- CNN LSTM
  - CNNs to learn features (e.g. images); LSTM for image sequences
- Encoder-Decoder LSTM
  - One LSTM network → encode input; one LSTM network → decode output
- Bidirectional LSTM
  - Input sequences are presented and learned both forward & backwards
- Generative LSTM
  - LSTMs learn the inherent structure relationship in input sequences; then generate new plausible sequences

Different Useful LSTM Models – Many other applications

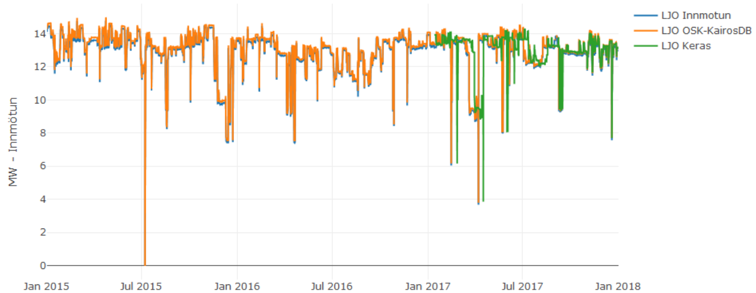
- Standard LSTM

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

```
# design network
model = Sequential()
model.add(LSTM(
    units=config['units'],
    input_shape=(train_X.shape[1], train_X.shape[2])
))
model.add(Dense(1, activation=config['activation']))

model.compile(loss=config['loss'], optimizer=config['optimizer'])

# fit network
print("Fitting model..")

history = model.fit(
    train_X,
    train_y,
    epochs=config['epochs'],
    batch_size=config['batchsize'],
    validation_data=(test_X, test_y),
    verbose=2,
    shuffle=config['shuffle']
)
```

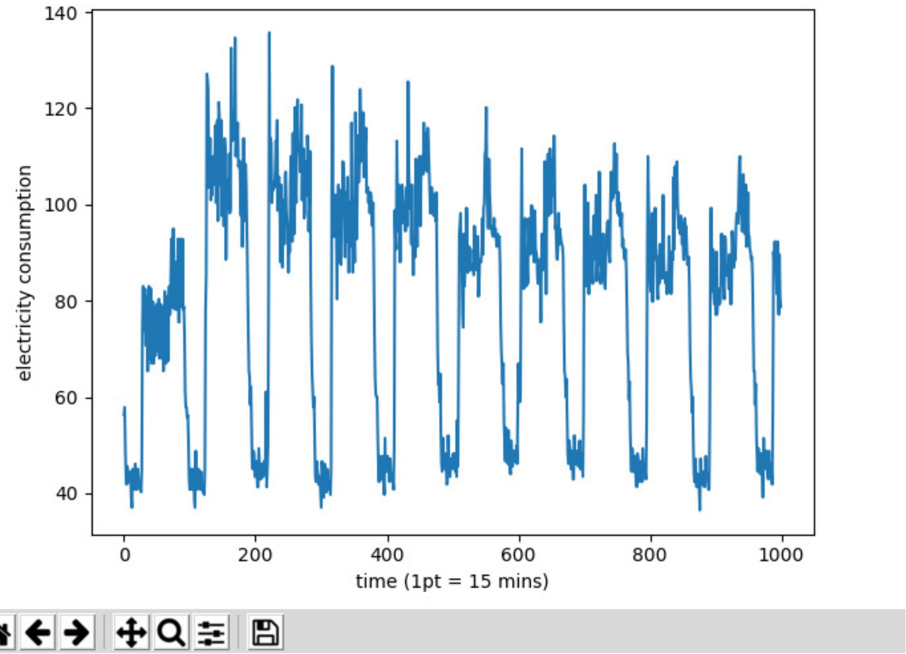
- **LSTM models work quite well to predict power but needs to be trained and tuned for different power stations**
- **Observing that some peaks can not be 'learned'**

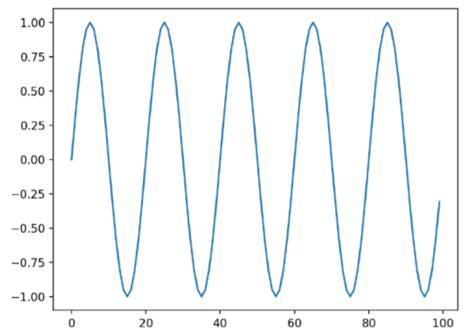
Landsvirkjun
National Power Company of Iceland



Different Useful LSTM Models – Stacked LSTMs

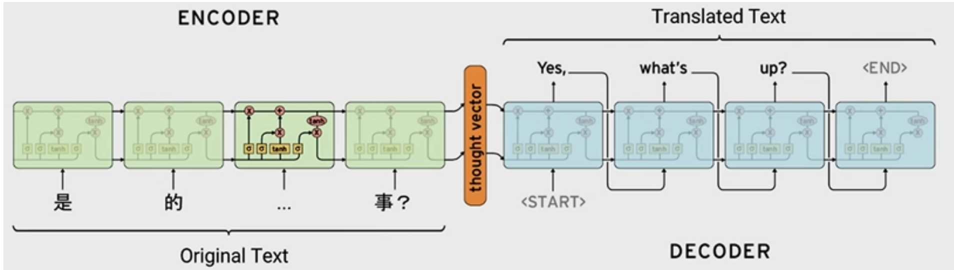
- E.g. predicting electricity consumption / customer
  - Stacked LSTM cells
  - Periodic elements can take advantage of state
  - Needs to be carefully tuned
  - Requires through use of state more computing
- E.g. damped sine wave prediction
  - Stacked LSTM cells since again periodic character
  - Depending on wave the pattern might be not able to be detected w/o LSTMs

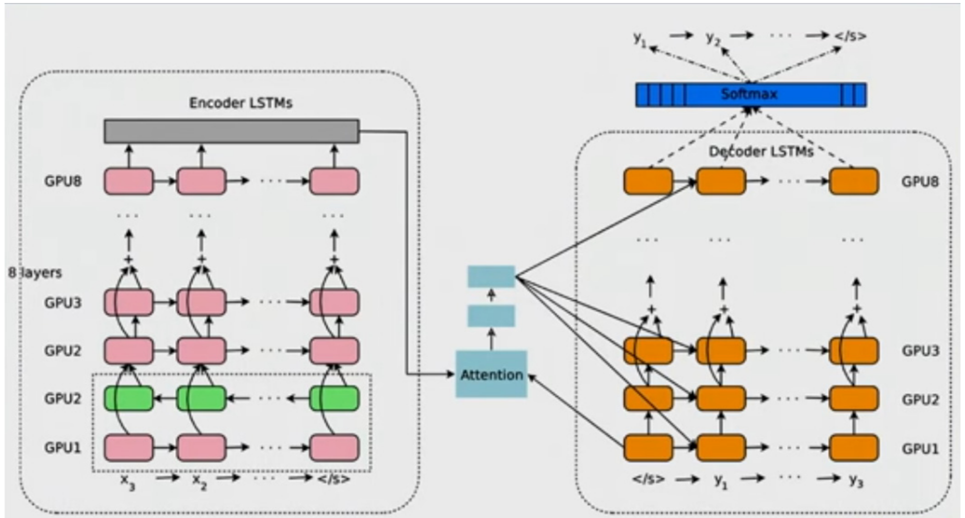




Tensorflow – LSTM Google Translate Example & GPUs

- Use of 2 LSTM networks in a stacked manner
  - Called ‘sequence-2-sequence‘ model
  - Encoder network
  - Decoder network
  - Needs context of sentence (memory) for translation

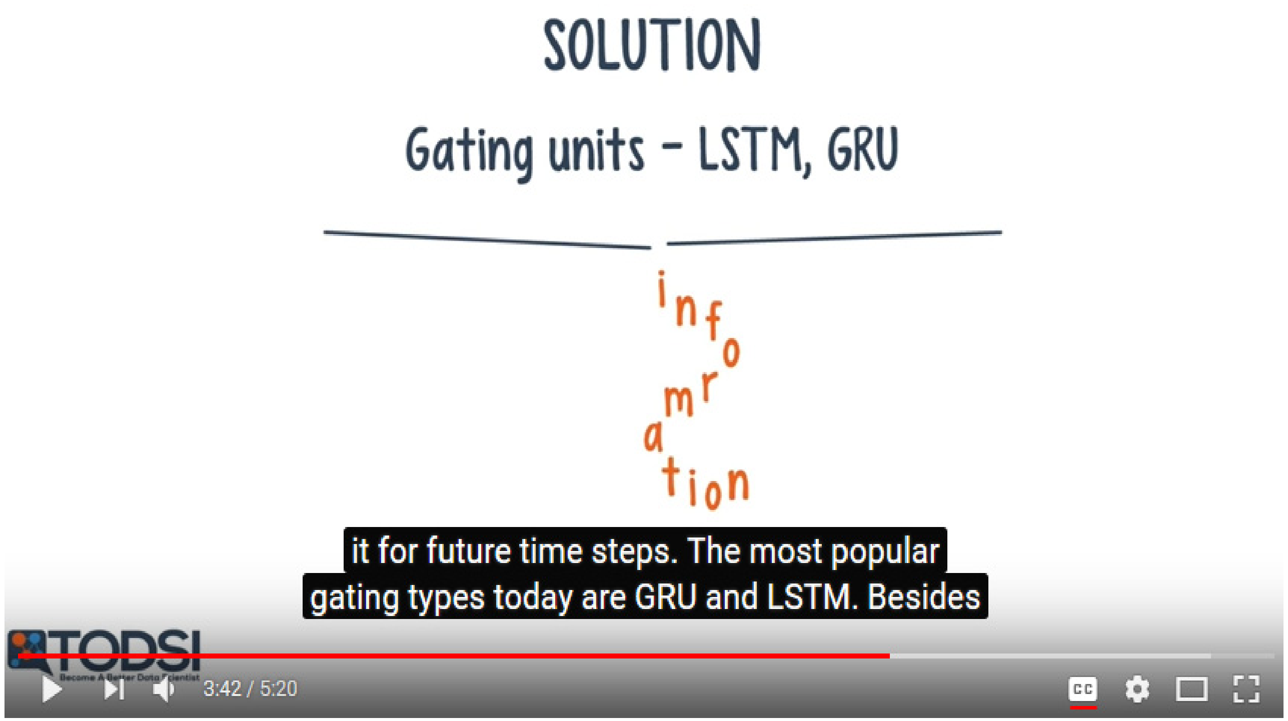




[5] Sequence Models

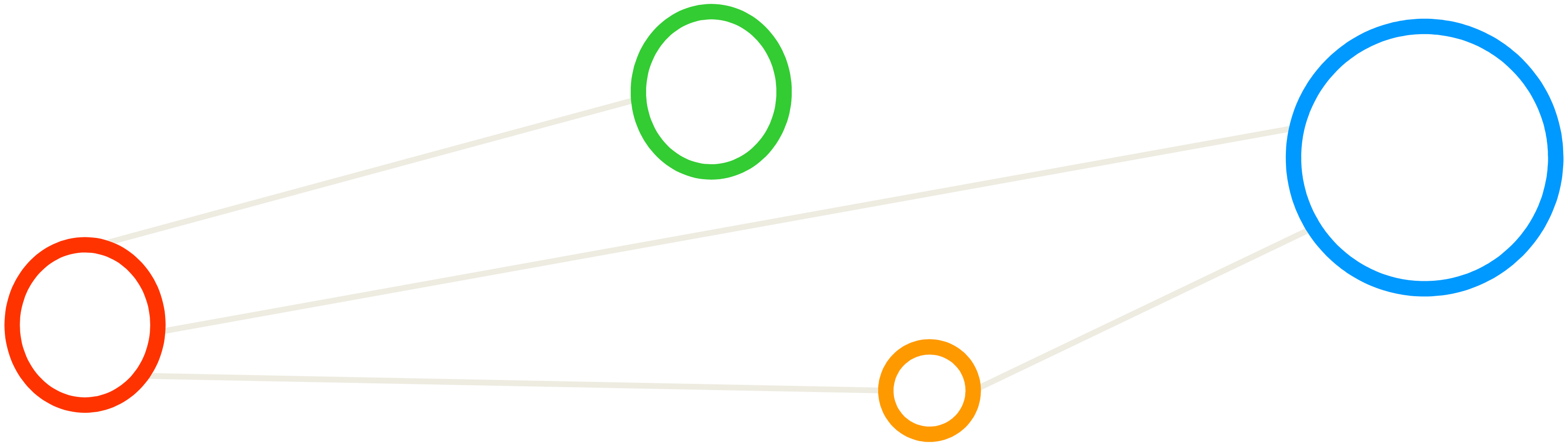
Exercises – LSTM Example – Revisit Group Outputs

[Video] RNN & LSTM



[6] Recurrent Neural Networks, YouTube

Lecture Bibliography



Lecture Bibliography (1)

- [1] Keras Python Deep Learning Library,
  Online: https://keras.io/
- [2] Theano Deep Learning Framework,
  Online: https://github.com/Theano/Theano
- [3] LSTM Networks for Sentiment Analysis,
  Online: http://deeplearning.net/tutorial/lstm.html
- [4] Tensorflow Deep Learning Framework,
  Online: https://www.tensorflow.org/
- [5] YouTube Video,'Sequence Models and the RNN API (TensorFlow Dev Summit 2017)',
  Online: https://www.youtube.com/watch?v=RIR_-Xlbp7s
- [6] YouTube Video, 'Recurrent Neural Networks - Ep. 9 (Deep Learning SIMPLIFIED)',
  Online: https://www.youtube.com/watch?v=_aCuOwF1ZjU&t=7s
- [7] Timeless Texts, Cutting-Edge Code: Free downloads of Shakespeare from Folger Digital Texts,
  Online: http://www.folgerdigitaltexts.org/download/
- [8] A. Gulli and S. Pal, 'Deep Learning with Keras', Packt Publishing,2017, ISBN 978-1-78712-842-2
- [9] O. Vinyals et al., 'Grammar as a Foreign Language', Advances in Neural Information Processing Systems, 2015, Online: https://arxiv.org/abs/1412.7449

Lecture Bibliography (2)

- [10] A. Karpathy and F. Li, ‘Deep Visual-Semantic Alignments for Generating Image Descriptions’, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, Online: https://cs.stanford.edu/people/karpathy/cvpr2015.pdf
- [11] R. Socher, ‘Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank’, Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP). Vol. 1631, 2013, Online: https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf
- [12] YouTube Video,‘RNNs‘, Online: https://www.youtube.com/watch?v=H3ciJF2eCJI
- [13] Kaggle, ‘UMICH SI650 – Sentiment Classification‘, Online: https://www.kaggle.com/c/si650winter11
- [14] Adventures in Machine Learning, Keras LSTM tutorial, Online: http://adventuresinmachinelearning.com/keras-lstm-tutorial/


services
measurement
within
fusion
new
funding
Policy-based
concepts
device
Analysis
forms
cross-disciplinary
resources
Centre
climate
Computer
expertise
Approaches
construction
Fusion
year
like
Computational
Test
E.g
Computing
PB
community
disciplines
results
Enable
Cross-Disciplinary
Modelling
different
EU
methods
High
big
nodes
Environment
Key
scientific
important
Science
centers
storage
basis
Infrastructure
files
etc
DKRZ
Use
simulations
ENES
technologies
model
formats
physical
SMAQ
international
engineering
References
Big
brain
increasing
via
EUDAT
processing
IPCC
Common
TB
Project
Data
Methods
MapReduce
create
computing
Using
European
databases
GB
Supercomputing
Work
images
codes
analysis
DLCL
stored
Resources
analyze
Large
Summary
data-intensive
Service
Scientific
levels
often
use
Level
Research
Matters
Concrete
external
performance
JSC
computational
research
e.g
science
HPC
large-scale
Juelich
system
datasets
acquisition
compute
Landscape
System
Climate
modelling
AR5
Hardware
distributed
cores
access
hundreds
Nutshell
Services
management
network
Technologies
Earth
Structure
current
DLCLs
Understanding
structures
directory
project
General
Germany
Simulation
Processing
Provide
NASA
Energy
systems
day
Health